HIGHFALUTIN' COMPUTIN'

with

Bob Orrfelt

on your

Timex Sinclair 1000™ Computer

Jim March

HIGHFALUTIN'

COMPUTIN'

by

Bob Orrfelt

1982

1st printing:
December 1982

Bob Orrfelt
(415) 369-9136

Table of Contents

LIST OF ILLUSTRATIONS

LIST OF TABLES

PROGRAM LIST

PREFACE

This book is the result of my experience with first a ZX81 computer and then a Timex Sinclair 1000 computer. The ZX81 is the product of Sinclair Ltd. in England. Except for a 2K instead of a 1K memory, the two are functionally the same. However, the 2K makes a world of difference. Most of 1K memory is used up for internal purposes or needed for the TV screen display. This book is written for a computer with 2K memory.

The purpose of this book is to look at the computer from as many viewpoints as possible. Topics include the beginning of BASIC, hardware, programming, internal files, and machine code. Much of the material is for the beginner. Some more advanced material is given as relates directly to the Timex Sinclair 1000 computer, for those who wish to pursue some (or all) of the topics.

There is a glossary of computer terms at the end of this book. Use it together with your User Manual furnished with your computer.

A number of other books on the ZX81 or Timex Sinclair 1000 were written in England and republished in the USA. One difference which may be confusing is the reference to NEWLINE. This term was changed to ENTER for the US models. It is the key called RETURN on most keyboards. If you are looking for more refernce material, you have an 8K ROM and 2K memory. (A Sinclair ZX80 had a 4K ROM). Many of the programs written for 16K memory (add-on) will work on your 2K computer.

The student of computer hardware will need digital logic design and Z80 microprocessor books. The information given in Chapter 2 should give you an idea of what is involved.

Beware of BASIC programming books not written for the Sinclair BASIC unless you intend to program on other computers. Most BASIC's use DATA and READ statements which the 1000 does not have. If you start with Sinclair BASIC you will find it easy to learn and easy to go on to other computers.

Lots of programs have been written for Sinclair BASIC. Look for programs written for the 8K ROM. There are many sources: books, magazines, and user clubs.

If you wish to get into machine code, get a book on the disassembled ZX81 8K ROM. Also you will need a book on the Z80 microprocessor assembly language.

Have a ball with your Timex Sinclair 1000 computer. You'll get out as much as you put in. Happy computin'.

FIRST TIME TURN-ON

The following procedure will check out your computer for proper operation.

1. Set-up your computer, switch box, and power cord as shown in your User Manual.

2. The inverse K (white on black square) letter shows that most of your computer is working.

3. To test the keyboard, press the P key and the word PRINT will appear. An inverse L will follow the word PRINT. This shows the computer is expecting letters to be typed in next.

4. Hold down the shift key and press the P key again. The result is a ".

5. Type in QWERT and another ". You should now have PRINT "QWERT" and an inverse L. This tests that all the keyboard lines are working.

6. Press the key marked ENTER. The QWERT will appear at the top of the screen.

7. Go to Chapter 3 of your User Manual and follow the examples. Learn to use the DELETE key (shifted 0).

8. If all functions are working, you are now ready to type in a program or do a calculation. Start computin´.

CHAPTER I

INTRODUCTION

SCOPE

This book is about the Timex (R) Sinclair (TM) 1000 computer (or simply the 1000). It is truly a terrific little machine with giant capabilities. It is of the class of general purpose computers which means it is easy to program and easy to add external equipment. You are not bound to ROM pack cartridges. The 1000 can be used to play games, learn BASIC programming, learn computer hardware, solve problems, or act as a controller. It is inexpensive enough to be dedicated to a single task. In the following chapter we will look at the computer from several different viewpoints.

ORGANIZATION OF THIS BOOK

This chapter is an overview of the 1000. It includes a brief physical description of the unit and a look at the development of the BASIC language.

Chapter II discusses the 1000 from the hardware viewpoint. This includes the TV set, power supply, cassette recorder, computer circuits, and the edge connector interface. An interface example with software is included.

Chapter III tells you how to operate your computer including writing menus for tape cassette files.

Chapter IV is on how to write programs using BASIC.

Chapter V talks about animation and game programs.

Chapter VI gets into how the computer works from the CPU to the way RAM files are organized.

Chapter VII is a hodgepodge of useful or just clever ideas.

Chapter VIII is an easy start on hexadecimal code.

PHYSICAL DESCRIPTION

The computer is about 6 3/8 inches wide, 6 7/8 inches deep, and 1 3/8 inches high at the rear. A two inch clearance is needed on the left side for electrical connections. If attachments are used, space will be needed to the rear. A 40 key pressure-sensitive membrane keyboard is about 6 inches wide and 2 inches deep. Connectors are:

```
9VDC - Use power supply  provided or see Chapter II.
MIC  - Jack for saving programs on cassette tape.
EAR  - Jack for loading programs from cassette tape.
TV   - RCA phono type jack for connection to TV.
No name - rear edge connection for add-ons.
```

A switch underneath selects either TV channel 2 or 3 for best reception. Inside is a Z80A CPU, 8K ROM with BASIC, 2K RAM and a special I/O i. c. for the keyboard, TV display, and the cassette interface.

Connecting cables for a TV set and for a cassette tape recorder are supplied.

BASIC

The BASIC language was developed at Dartmouth College in 1965 under a grant from the National Science Foundation. The work was directed by Professors John G Kemeny and Thomas E. Kurtz. BASIC was popularized by General Electric time-sharing service. This original BASIC was designed to provide numerical answers to numerical problems. The name BASIC is an acronym for Beginner's All-purpose Symbolic Instruction Code.

Since the time-sharing service used a teletype, BASIC used capital letters. Each line was called a statement. Lines were numbered for editing. A typical line would be:

```
0010 LET X=5A
```

Line numbers were usually multiples of 10 so that lines could be inserted if needed later. The statement began with an English word followed by an algebraic expression.

In the example, X is the unknown numeric variable, 5 is a constant, and A a given numeric variable. At first only a few abbreviations crept in, such as REM for remark, DIM for dimension, and GOSUB for "go to a subroutine". As the primary input and output device was the teletype, the use of the word PRINT was appropriate for printing messages.

A few years later, string variables were added to BASIC. A $ is used to denote a string of characters. Words like STR$ and CHR$ were added to the BASIC language. A string variable can be handled by algebraic functions the same as numeric variables. This enables BASIC programs to sort a list of names, compare words, etc. At this point in time, the language became useful for many applications beyond just solving numeric problems.

As BASIC was improved and adapted for various computers, many dialects were developed. The fundamental algebraics were kept but many extensions, and formatting, file, and editing commands were added. Usually file and editing commands are dependent on the computer hardware. This makes it difficult to adapt a BASIC program from one computer to another. However, the idea can often be adapted.

The change from teletype to a TV display changed PRINT to mean "show on the TV set". If you have a printer, LPRINT will do the printing. LPRINT stands for line printer and these are only used on giant computers. So the BASIC language becomes richer and more difficult to learn.

The American National Standards Institute has published a proposed "minimal BASIC". Table 1-1 is a comparison with 1000 BASIC. The 1000 has dropped the statements DATA, READ, and RESTORE. Programmers who have learned other BASICs probably will have great difficulty writing a program without these statements. New programmers won't miss them at all. The original idea goes back to batch processing where data was fed into large computers for processing. The advent of real time personal computers has diminished the need for these statements.

A computer program must be converted to machine code before it can be acted on. The two ways are called "interpreted" and "compiled". The 1000 is mostly interpreted, which means the program is converted to machine code line-by-line as it is run. Compiled would mean the entire program was converted and then run. Interpreted languages are easier to use but do not run as fast as compiled languages. Most of the original algebraic words and functions together with string variables are used by the 1000.

Your 1000 computer has many extensions to the original BASIC language. These include PEEK, POKE, and USR which permit direct access to the internal workings and machine code. Numerical names may be of any length, strings may be of any length, the line length of a statement may be of any length. Numbers may be in the range of + or - 3X10E-39 to + or - 7X10E38 to 9 1/2 decimal digits. FOR/NEXT loops may be nested 26 deep. All BASIC words, functions, and commands are marked on the keyboard.

Table 1-1. Minimal BASIC/1000 BASIC

Proposed ANSI	1000	Comment
DATA	(not used)	Use LET or GOSUB
DEF	(not used)	Use GOSUB
END	(not used)	Not needed
FOR/NEXT	FOR/NEXT	
GOSUB	GOSUB	
GOTO	GOTO	
IF/THEN	IF/THEN	
INPUT	INPUT	
LET	LET	
ON/GOTO	(not used)	Use IF/THEN GOTO
OPTION BASE	(not used)	
PRINT	PRINT	
RANDOMIZE	RAND	
READ	(not used)	See DATA
REM	REM	
RESTORE	(not used)	See DATA
RETURN	RETURN	
STOP	STOP	

Functions: TAB, ABS, ATN, COS, EXP, INT, LOG (use LN), RND, SIN, SQR, and TAN are common.

CHAPTER II

GETTING ORGANIZED

This chapter discusses the 1000 from the hardware viewpoint. Included are the TV set, power supply, cassette recorder, computer circuits, and edge connector interface.

TV SET

Black and white sets give a better display than color, as black and white sets have a wider bandwidth which results in clearer characters. The color on color sets tend to smear the dots that make up the display. This is true for all computers, not just the 1000. The most popular size is a 12 inch set. Readjusting TV tuning, brightness, and contrast controls can improve the display.

POWER SUPPLY

You will normally use the power supply that comes with your computer. If for some reason you wish to use your own, the specifications are: +9 vdc (7 to 11) well filtered. The tip of the 3.5 mm power supply plug is positive. The power unit supplied has a rating of 1 ampere so you have extra power for line fluctuations, temperature changes, and added equipment.

CASSETTE RECORDER

Again, simpler is better. Select an inexpensive cassette recorder such as a GE Model No. 3-5009A. Battery operation eliminates hum from the power supply. About full volume should give the right level for SAVEing programs and about 10 15% less for LOADing programs from tape. The signal is about 3.2 kHz. A "1" is eight cycles and a "0" is four cycles. Bass should be reduced (low level) if possible. The "MIC" output is about 0.001 vrms designed for the microphone input on a cassette recorder. The "EAR" input is expecting about 1 v rms (about 4 volts peak-to-peak) which is a common cassette output for 8 ohm earphones.

With two cassette recorders you can duplicate tape using the simple circuit shown in Figure 2-1. Recommended is C-10 tape cassettes which record about 5 minutes per side. One side will hold all of the programs in the Appendix at the end of this book.

ACCESS

Why do you need access to the inside of your computer? You don't. However there are other people who are going to change things. For those other people there are some things they should know.

NOTE

The warranty is void if the computer has been tampered with.

There are NO user serviceable parts inside.

Now that you have been warned we can proceed. Figure 2-2, bottom view, shows the locations of the five screws holding with an adhesive. Remove these screws and the bottom cover for access.

CAUTION

The keyboard connector strips are easily torn and are not repairable.

The next step is critical. Remove the two screws shown in Figure 2-2, bottom view with cover removed, while holding the pc board in place. Whith great care, raise the pc board and work the keyboard connector strips from their sockets. The computer is now disassembled.

COMPUTER MONITOR

A computer monitor gives better resolution and those with green phosphor are easier on the eyes. The circuit shown

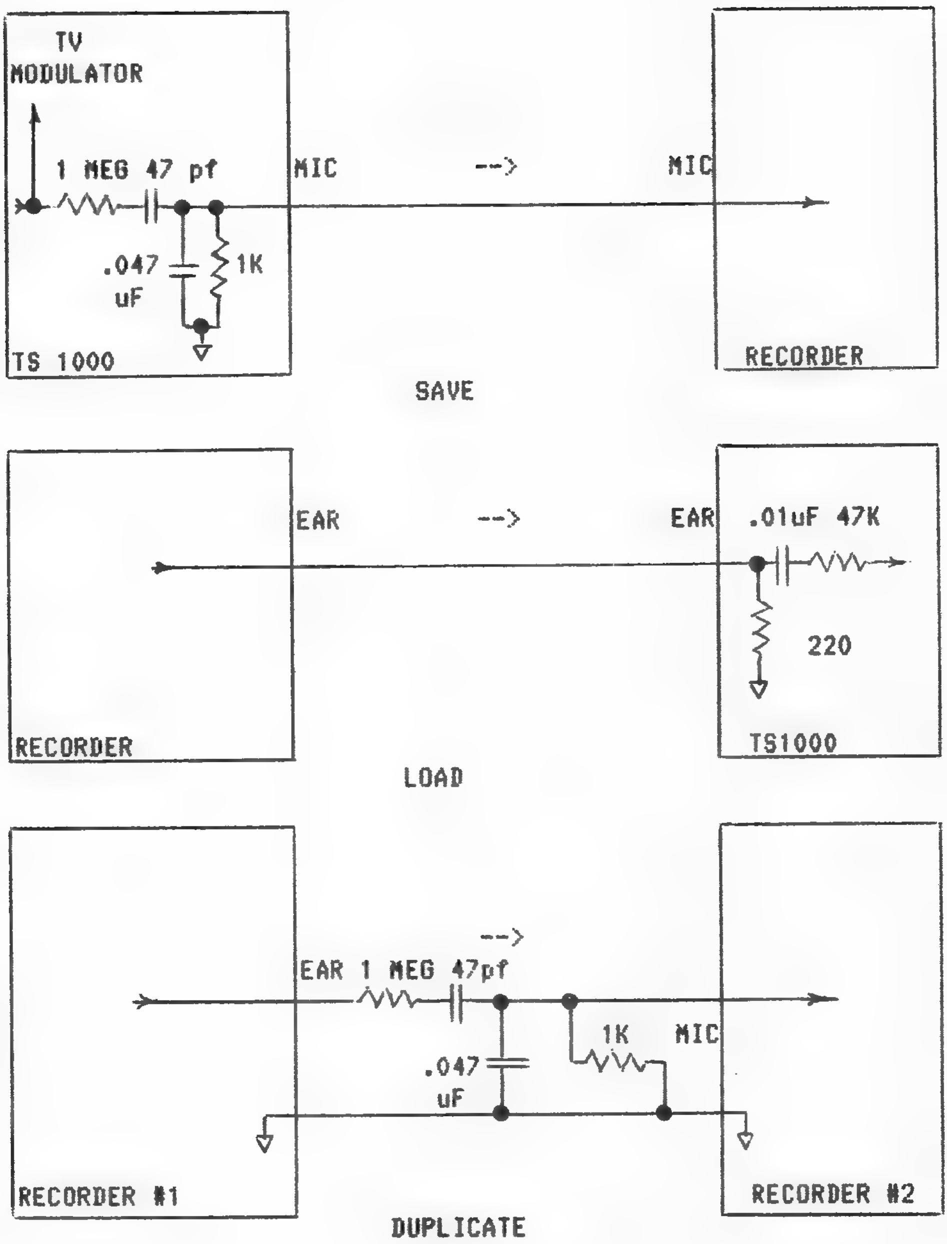
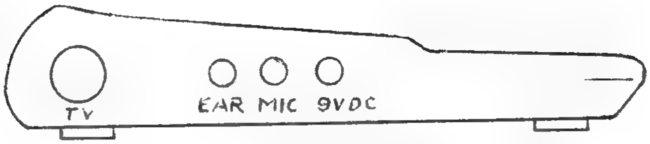
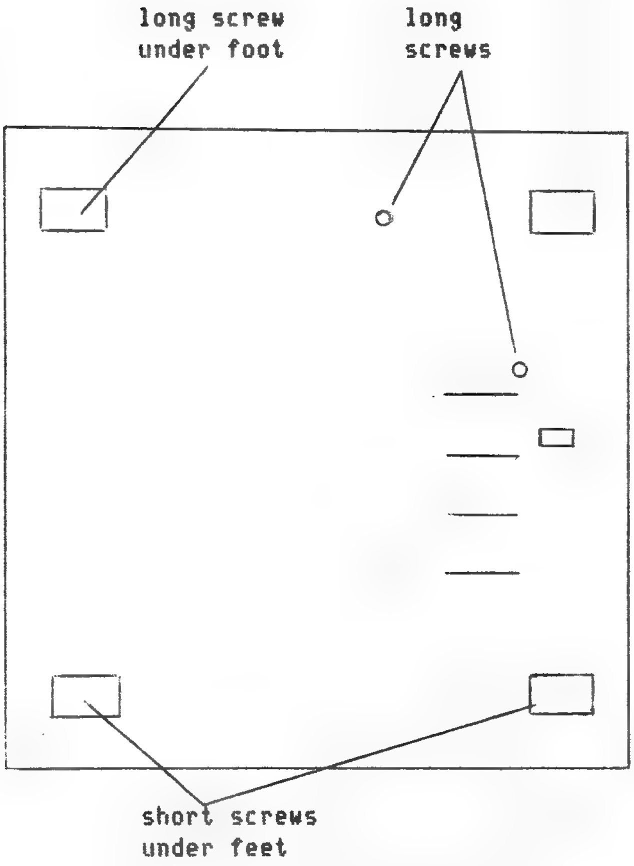


Figure 2-1. Mass Storage



Side View



Bottom View

Figure 2-2a. Internal 1000 Access

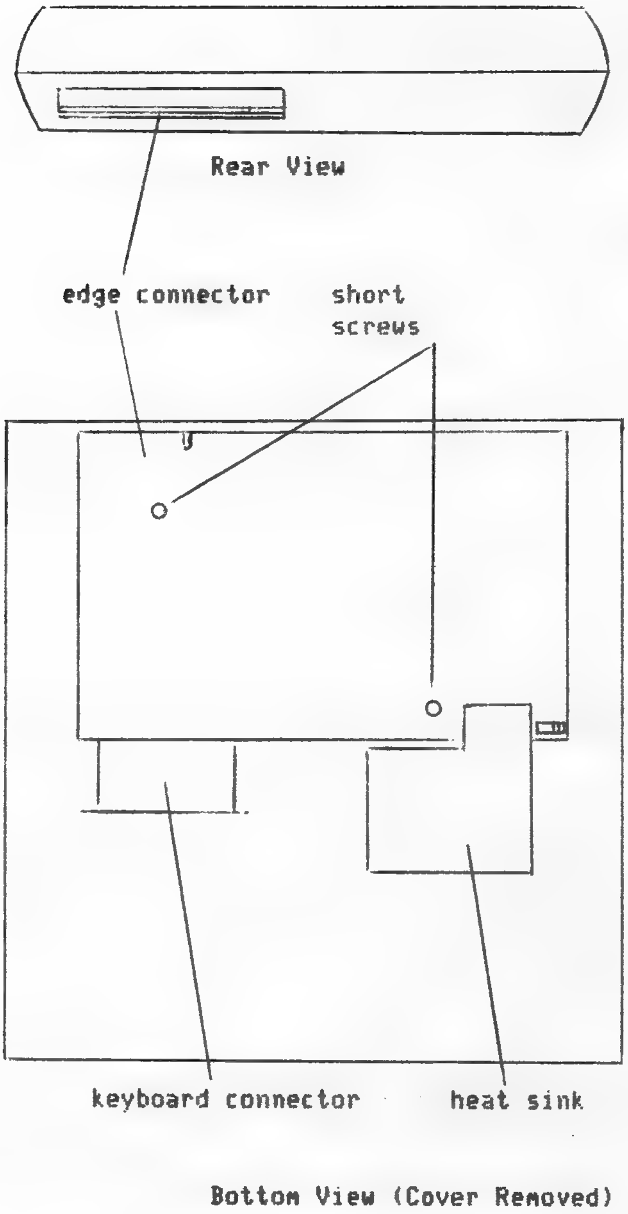


Figure 2-2b. Internal 1000 Access

in Figure 2-3 can be installed. USA3 ~~UK1~~ is between the EAR jack and the ASTEC modulator. C4 is between +5 vdc and ground. These can be used for the power connections.

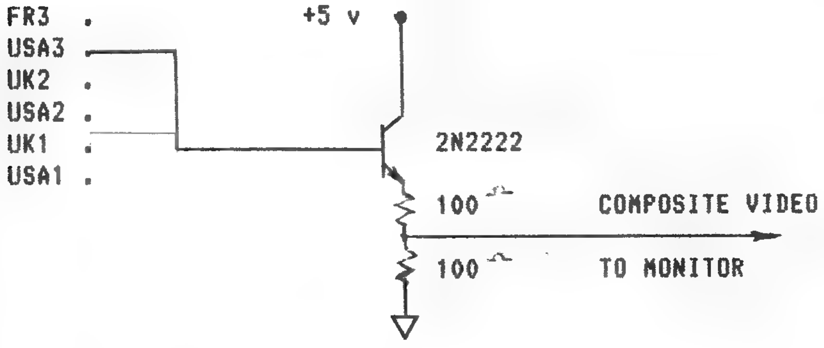


Figure 2-3. Monitor Interface.

KEYBOARD

There are those who wish a full size keyboard (surely not you). You must use a keyboard with 40 switches of the simple switch type. The keyboard must be unencoded and wired as shown in Figure 2-4. The pull-up resistors and diodes shown are already on the 1000 circuit board. The two connectors KB1 and KB2 are clearly morked on the 1000 circuit board.

The pull-up resistors normally keep the lines going to IC1 at a logic high. When a key is pressed and an address line A8-A15 goes low, one of the lines to IC1 is pulled low. The key is thus identified and encoded. This code goes over the DATA lines to the CPU.

You'll probably have some extra keys on your keyboard. One function you might add is two wires, one to the reset pin of the Z80 and the other to ground. This key will reboot the system without your having to pull the power plug.

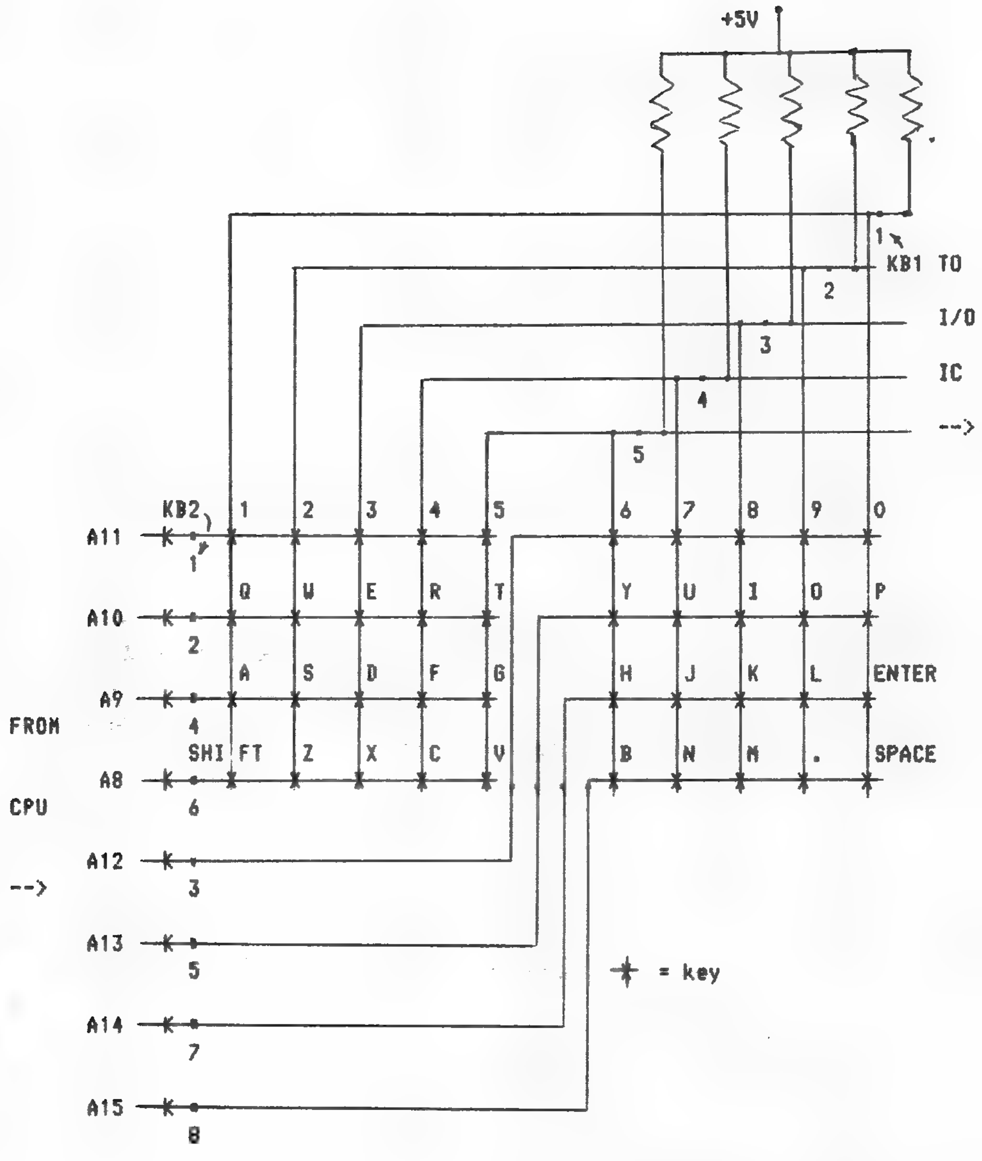


Figure 2-4. Keyboard Matrix.

MEMORY

The memory chip used for RAM is TMM2016P-1. A 6116 2K by 8 bit memory chip works just as well. If you buy a 16K RAM pack, the 2K chip on the board is not used. It is locked out by the RAM putting +5 volts on the RAMCS (RAM chip select) line to the 2K RAM.

REGULATOR

About the only other change might be to remove the +5 volt regulator if you have an external regulated power supply plugged-in to the rear edge connector.

COMPUTER CIRCUITS

The following discussion leads to the use of the rear edge connector.

Z80 CPU

The Z80A CPU microprocessor has 16 address bus lines (A0 through A15), 8 data bus lines, and a number of control lines. Of interest here are the following:

Address bus:	Used to address ROM and RAM
	Used for dynamic memory refresh
	Used for keyboard scanning
Data bus:	RAM and ROM data
	Keyboard data
	Tape data in
	Tape/TV data out
Control lines:	IOREQ - Input/Output Request
	MREQ - Memory Request
	RD - Read
	WR - Write

All this is fairly standard except for using address bus lines to scan the keyboard. Not obvious is that the ROM/RAM chip selects are not fully decoded as A15 is not used.

IC1 LOGIC CHIP

The IC1 logic chip acts as a multiple input/output port. Four functions are performed:

TV/Tape output parallel-to-serial conversion
Tape input serial-to-parallel conversion
Keyboard encoder
ROM/RAM chip select encoder

The first three functions use the I/O port functions of the Z80A CPU. The TV scan uses the non-maskable interrupt and the keyboard uses the maskable interrupt. The other way of getting data in and out of the computer is by using memory addresses. The decode for ROM chip select uses control lines MREQ and RD. The decode for the internal RAM chip select uses control lines MREQ and either RD or WR. IC1 only uses address line A14 for decode. If A14 is low then the ROMCS line is active and if high the the RAMCS line is active. This means that for the lower 16K addresses and the 16K above 32K the ROM is selected. The ROM is only 8K so if you do a 65K memory dump you get the ROM four times. The RAM is only 2K and so you will select it 16 times. You can PEEK at any memory address and one of the chips is selected. RAMCS and ROMCS lines do have buffering resistors so that an external device can prevent lines from going low (true).

EDGE CONNECTOR INTERFACE EXAMPLE

Let's design a circuit so you can turn your cassette player on and off using BASIC commands. This means you can load programs under program control. First we need the hardware and then the software.

The cassette recorder has a switch input for starting and stopping the cassette. Often the switch is on a hand held microphone. When we plug in a microphone switch plug, the circuit is broken and the cassette stops. Checking a little further we find the shell of the plug goes to the battery and the tip to the motor. Battery voltage under load is about 6 volts dc.

Table 2-1. Rear Edge Connector

Pin No.	Name	Description	Logic Level
1A	D7	Data line #7	High true
2A	RAMCS	RAM chip select from I/O chip	Low true
3A	-	Keyway slot	-
4A	D0	Data line #0	High true
5A	D1	Data line #1	High true
6A	D2	Data line #2	High true
7A	D6	Data line #6	High true
8A	D5	Data line #5	High true
9A	D3	Data line #3	High true
10A	D4	Data line #4	High true
11A	INT	Interrupt request	Low true
12A	NMI	Non-maskable interrupt (RST 66)	Low true
13A	HALT	Halt state (output)	Low true
14A	MREQ	Memory Request (output)	Low true
15A	IOREQ	Input/Output Request (output)	Low true
16A	RD	Memory Read (output)	Low true
17A	WR	Memory Write (output)	Low true
18A	BUSAK	Bus Acknowledge (output)	Low true
19A	WAIT	Wait state (input)	Low true
20A	BUSRQ	Bus Request (input)	Low true
21A	RESET	Reset CPU (input)	Low true
22A	M1	Machine cycle one (output)	Low true
23A	RFSH	Memory refresh (output)	Low true
1B	5V	5 volt power supply	-
2B	9V	9 volt power supply	-
3B	-	Keyway slot	-
4B	0V	Power supply return	-
5B	0V	Power supply return	-
6B		Clock phase 0	-
7B	A0	Address line #0	High true
8B	A1	Address line #1	High true
9B	A2	Address line #2	High true
10B	A3	Address line #3	High true
11B	A15	Address line #15	High true
12B	A14	Address line #14	High true
13B	A13	Address line #13	High true
14B	A12	Address line #12	High true
15B	A11	Address line #11	High true
16B	A10	Address line #10	High true
17B	A9	Address line #9	High true
18B	A8	Address line #8	High true
19B	A7	Address line #7	High true
20B	A6	Address line #6	High true
21B	A5	Address line #5	High true
22B	A4	Address line #4	High true
23B	ROMCS	ROM chip select from I/O chip	Low true

Let's develop the circuit shown in Figure 2-5. To operate the relay we can use a TIP 110 darlington power transistor. A 511 ohm resistor turns it on. And now we need a circuit to turn it off.

We'll want to latch the cassette on (and off) so we can LOAD a program. A simple latch is a 7474 I.C. having a PRESET and CLEAR suitable for our purpose. Either the Q or NOT Q output can cut off our transistor. Now we will look at the computer.

All of the CPU lines seem quite busy. The port IOREQ is used for the keyboard scan, A0 through A14 for memory, all address lines for the keyboard and dynamic refresh. A combination not used is MREQ, WR, and A15. We need a data line to use POKE so let's pick D0.

We have four lines in from the computer and need these decoded to PRESET and CLEAR our 7474. Fortunately a 7442 will fill the bill. We assign the four inputs and refer to an I.C. handbook for the truth table and find the circuit shown in Figure 2-5.

We did not inhibit the RAMCS so we need an address to POKE that will do no harm.

We now need software to operate the above circuit. There is an unused byte in RAM at address 16507 (see Table 8-2). If we add 32768 (2 to the 15th power) we get address 49275. If we POKE 49275,1 the byte is put in RAM at 16507 and resets the 7474 at the same time. POKE 49275,0 will set the 7474 in a similar manner.

Here is how to load one program from another. The first program will have the following lines.

```
        1000 POKE 49275,0   [turns on recorder]
        1010 LOAD "2"
```

The second program will have the following lines:

```
        400 SAVE "2"
        410 POKE 49295,1  [turns off recorder]
        420 RUN
```

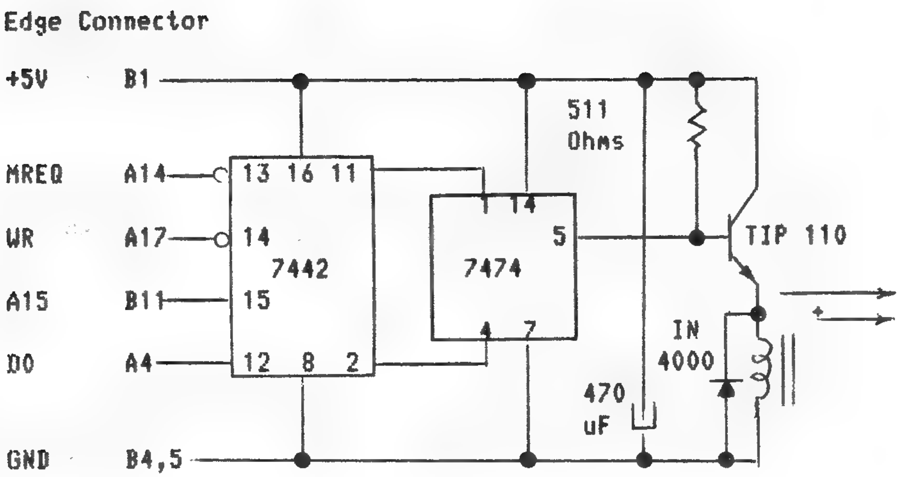


Figure 2-5. Auto Cassette Circuit

When the first program gets to line 1000 the recorder is turned on and the second program will be loaded. The second program will automatically be run. If you wish the second program can be made to load program #3 and so on.

For the above circuit, Radio Shack 275-246 relay was used. The relay contacts are rated at 115VAC at 3 amps. This circuit can be used to control other devices within this rating.

CHAPTER III

OPERATION

This chapter is about operating the 1000 as distinct from BASIC programming. The only way to learn operation of any computer is by having lots of hands on (or fingers on) practice. The Appendix at the end of this book is intended as an exercise for this purpose. Once the operation of the keyboard and cassette recorder has been learned, BASIC programming will be much easier.

Whenever the computer is waiting for you, its status is shown by a prompt or a message at the lower left corner of the screen. There are four prompt characters: an inverse K, inverse L, inverse F, and an inverse G. Messages have two parts: either two numbers or a letter and a number. The two parts are separated by a slash (/).

THE "K" PROMPT

When the computer is first turned on, the ROM program puts the computer in an empty state waiting to be told what to do. The keyboard is being scanned to determine if any key has been pressed. The program presumes the key to indicate a keyword or a line number in a BASIC program. Spaces can be entered but are ignored. There are two kinds of keywords: commands and BASIC words. FAST, SLOW, LIST, and CLEAR are examples of commands. GOTO, LET, INPUT, and NEXT are examples of BASIC words. When the ENTER key is pressed, the line that has been typed is checked by the program for syntax (format) and if an error is detected, an inverse S is placed on the screen. If all is OK, the command is performed, the line is entered into the program, or the BASIC definition is placed in memory.

After a keyword has been entered the prompt is changed to an inverse L indicating a letter is expected.

THE "L" PROMPT

A letter includes 0 through 9, the alphabet, symbols, and punctuation. Historically the first computers were number crunchers and as a result, letters of the alphabet are used to indicate variables (numbers). If you really mean a letter of the alphabet it is indicated by placing the letters in quotes. Therefore, PRINT B means print the number represented by B. Print "B" means print the letter B. "B" or "A BUNCH OF LETTERS" are called strings. Some BASIC words can be entered in the inverse L mode such as AND, THEN, TO, and STEP. The word THEN returns the input to the keyword mode. For example: IF N=5 THEN GOTO 10.

The inverse L is also used as a prompt while running a program. If just the L is displayed, it indicates a number is expected, or if it is in quotes it indicates a string is expected. If you wish to STOP the program, eliminate the first quote with the DELETE key and enter STOP. The second quote is ignored.

THE "G" PROMPT

This prompt indicates the graphic character mode. You get into the graphic mode by holding down the shift and pressing the 9 key. All of the alphabet, numbers, and puntuation are displayed as inverse video characters. The space is displayed as a black square. In addition there are 21 special graphic characters as shown on the keyboard. In this book graphic characters are shown as [gH] for graphic H and [gsH] for a graphic shifted H. If you are in the grahpics mode, you can press the graphics key or the shifted ENTER key to return to the letter input mode.

THE "F" PROMPT

To enter one of the BASIC functions (SIN,COS, STR$, etc.) use shifted ENTER key. After the function is entered the mode is automatically shifted back to the letter input mode.

THE "S" PROMPT

When ENTER is pressed, an inverse S may appear somewhere in the line you just entered. This indicates that the programmer who wrote the BASIC language program didn't plan for what you typed. Use the editor to redo the line.

MESSAGES

When the computer completes a program - successfully or unsuccessfully - a report or message is given on the lower left part of the screen. These are either a letter and a number or two numbers.

THE EDITOR

The 1000 8K ROM provides what is known as a line editor. A portion of RAM memory is reserved for an input line and is called the ENTER file. This file contains the input line with keywords stored as one byte tokens while the display has the words all spelled out. Editing takes place in the ENTER file although it appears you are editing the display. So what you see is only an image of the line you are editing. You type into the ENTER file or you can bring an existing line from your program into the file.

To Correct a Line at the Bottom of the Screen

A prompt character is inserted as a cursor to indicate where the next typed character will be entered. If the DELETE key (shifted 0) is pressed, the chatacter to the left of the cursor is erased. The shifted 5 and shifted 8 move the cursor to the left and right without erasing characters. If the cursor is between characters, any characters you type will be inserted at that location.

To Change a Line in an Existing Program

The keyword EDIT (shifted 1) brings the "current" line into the ENTER file for editing. The current line is indicated in the program display by an inverse > after a line number. If the line has been deleted it will bring the following line down. To change the current line a line or two, use the shifted 6 or shifted 7 key. For larger changes, use LIST nnnn where nnnn is the line number you want.

To Replace a Line

If a line is entered with an existing line number, the old line will be replaced by the new line. This works with any line, not just the current line.

To Delete a Line

If you type in only a line number and press ENTER, if the line exists the line will be deleted including the line number. If you type in a new line number by itself and press ENTER the program file is unchanged and the number is discarded.

One more comment on editing. The program display will be slowed if you have been running a program with a lot of variables. To speed-up listing the display, press CLEAR to get rid of the variables file.

TAPE STORAGE

Before spending a lot of time entering long BASIC programs you should practice saving and loading programs using your cassette. If you have purchased software or received software from a club or friend, make a duplicate on a new tape. Otherwise use the programs given in the Appendix of this book. Digital recordings on cassettes are not highly reliable. You may wish to make more than one copy on the same or different tapes.

TAPE FILES

Use the following procedure to verify the operation of your cassette recorder. In this book, "type" means to type in a line but do not press the ENTER key. "ENTER" means to type in the line and to press the ENTER key. If your cassette has automatic level control, probably full volume works best for record and 10% less for playback. Proceed to:

1. ENTER the following program:
```
    10 REM "ONE"
    20 PRINT "ONE LOADED"
```

2. Type the following:
   SAVE "ONE"
3. Start the cassette in the RECORD mode (usually both the RECORD and PLAY keys). Press the ENTER key.
4. When the display has stopped jumping around, stop and rewind the tape.
5. ENTER the command NEW (above the A key).
6. Type the following:
   LOAD "ONE"
7. Put the tape in the PLAY mode and press ENTER. There is about a 6 second lead-in tone before the program. Press the ENTER key before the tone is up.
8. When the display stops jumping, stop the tape.
9. ENTER the RUN command (on the R key). The message "ONE LOADED" should be PRINTed on the screen. IF not, readjust the record or playback level.

THE MENU

A menu is a list displayed on the TV showing programs for your selection. If a menu program is used, it is the first program on a cassette tape. It has two functions:

1. Tells you the programs on the cassette.
2. Lets you select a program by number.

Program 3-1 shows the structure of a menu. The key idea is line 210 which loads the desired program from the cassette. Add a name, a list of the programs, a keyboard input, and an auto RUN feature and we have our program.

Program 3-1. A Simple Menu Program

```
 10 REM "0"
100 PRINT "1  Name of 1st program"
110 PRINT "2  Name of 2nd program"
120 PRINT "3  Name of 3rd program"
200 INPUT A$
210 IF VAL A$<1 OR VAL A$>3 THEN GOTO 200
220 LOAD A$
230 GOTO 10000
300 SAVE "0"
310 GOTO 100
```

The four parts to the above program are:

1. Line 10 is required to save the menu on cassette. The name "0" was chosen as the other programs will be named "1", "2", and "3" for loading by selection.
2. Lines 100-120 display the list for your selection.
3. Lines 200-230 get the program name and do the load. Line 210 checks for input error and line 220 LOADs the selected program. The program should never get to line 230 which stops the program.
4. Lines 300-310 are the auto RUN routine. This causes the program to RUN as soon as it is LOADed.

SAVING THE MENU ON CASSETTE

Once you have the menu typed in, the next step is to SAVE it on cassette. The SAVE command is on line 300 so we use GOTO 300.

1. Type GOTO 300
2. Start the cassette in the RECORD mode.
3. Press ENTER (within 6 seconds).
4. Stop the cassette. The tape is now in position for the first program to be recorded.

SAVING THE PROGRAMS

Now that we have the menu on cassette, the next step is to save some programs. Use NEW to erase the old program and type in a program. The name on the first line of the first program must be REM "1". This is used to SAVE and LOAD from the menu. After the program has been typed in:

1. Type SAVE "1"
2. Start the cassette in the record mode.
3. Press the ENTER key.
4. When the normal raster returns, stop the cassette.
5. SAVE the other programs using REM "2", REM "3", etc.
6. Rewind the cassette and load the menu (LOAD "" will always load the next program.
7. Type the program number from the menu selection.
8. Start the cassette in the play mode and press ENTER.

CHAPTER IV

HOW TO WRITE PROGRAMS

A computer program needs an input, an output, and something in between. In this chapter we will build a program (an adding machine), look at rules of programming, and give some program examples. Their are many ways of writing programs. Often we know what we have for input and what we want for output. What we need is the in between part which is the key idea to programming. This key is often a formula and once we have it we can just add inputs and outputs.

To get information into the computer we have two keywords LET and INPUT; to get it out we have the keyword PRINT. Consider this simple program:

```
10 INPUT BALANCE
20 INPUT AMOUNT
30 LET BALANCE=BALANCE+AMOUNT
40 PRINT AMOUNT,BALANCE
50 GOTO 20
RUN
```

As was stated in Chapter I, BASIC was originally designed to use numeric data to get numeric answers. The above is such a BASIC program. Let's examine its parts:

1. Line numbers show the sequence for running the program and are used for editing.
2. The key idea or formula is line 30 which reads "let the current balance equal the old balance plus the amount" (which may be negative).
3. Lines 10 and 20 let numbers be entered from the keyboard when the program is RUN.
4. Line 40 displays the results. The comma causes the display to be two columns.
5. Line 50 gets another amount. To STOP the program, just ENTER the keyword STOP.

If we use a B instead of the word BALANCE and an A instead of AMOUNT line 30 looks more like a formula: LET B=B+A. The choice is up to you.

NUMBERS

Now is the time to look at numbers in detail. All numbers are stored (and therefore remembered) to 8 digit precision positive or negative, and with a decimal place. When you write in a program: LET A=1 and LET B=A, the B is equal to 1. All INPUT and LET statements must evaluate to a number. There are two built-in numbers: PI and RND. RND stands for random and has a value of one of a sequence of 65536 numbers. Use of the keyword RAND (stands for randomize) makes RND a random number.

FORMULAS

Almost anywhere you can use a number you can use a formula if enough information already exists for the computer to reduce the formula to a number. BASIC can perform many types of calculations. This includes addition, subtraction, multiplication, division, square roots, raise a number to a power, and evaluate trigonometric functions as follows:

Symbol	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Raise to the power (X**2 means X squared)
SIN	Sine of a number in radians
COS	Cosine of a number in radians
TAN	Tangent of a number in radians
ARCSIN	Arcsine in radians
ARCCOS	Arccosine in radians
ARCTAN	Arctangent in radians
EXP	e to a power
LN	Natural logarithm
ABS	Absolute value
SQR	Square root
SGN	Signum (sign +1, -1, or 0)
INT	Integer (rounds down to whole number)

Parentheses are used to group those parts of a formula we wish to evaluate first. For example:

1. A formula inside parentheses is computed before being used in further computations.
2. Otherwise, the computer first raises a number to a power, then performs multiplication and division, and then addition and subtraction last.
3. Without parentheses, multiplication and division are performed from left to right and then addition and subtraction are performed from left to right.

If there is any doubt about the priority, use parentheses to eliminate any possible ambiguities.

STRINGS

In BASIC any sequence of characters enclosed in quotes is called a string. A string can be used after the keyword PRINT to prompt for inputs or for table headings, etc. In our adding machine program, we know how it works but maybe a user does not. We can add messages and headings to help a user. Try this:

```
100 PRINT "ENTER STARTING BALANCE";
110 INPUT BALANCE
120 PRINT BALANCE
130 PRINT "ENTER AMOUNTS ONE AT A TIME"
140 PRINT
150 PRINT "AMOUNT","BALANCE"
160 PRINT
170 INPUT AMOUNT
180 LET BALANCE=BALANCE+AMOUNT
190 PRINT AMOUNT,BALANCE
200 GOTO 170
```

Here we prompt for the two inputs and on line 150 we PRINT the heading. A PRINT by itself puts in a blank line. The semi-colon at the end of line 100 holds the printing position so that the balance entered next will be displayed on the same line.

STRING VARIABLES

A big step forward in making BASIC flexible and a big step backwards in readability was the addition of string variables. Strings are treated much like numeric variables in that they can be defined, stored, changed, and manipulated. A string is defined by using a LET statement and calling the string by a letter and a $ i.e. A$.

Well, A$ is not too readable (you can't use AMOUNT$) but let's see what else was added related to strings. You can add two strings of characters (BUG+OFF=BUGOFF) but how can you divide or multiply? You can't. There are some numbers associated with a string. Each character has a code which is a number. Also the number of characters in a string is a number. And if the character is a number from 0 to 9 it has a value. So three more words were added to BASIC. These three words result in a number and so can be used anywhere a number can be used.

Here are the three words:

CODE	[CODE "5" means 33 (the code for 5)]
LEN	[LEN "5" means 1 (the length i.e. number of characters in the string)]
VAL	[VAL "5" means 5 (the value of the character 5)]

The above gets you a number from a string. For the other way around you get a string from a number by using:

CHR$	[CHR$ 33 means "5"]
STR$	[STR$ 33 means "33"]

These five words are usually used with variables such as:

```
A$="PAGE "+STR$ P
```

These are all good words but are learned by needing them and by using them. Let's get back to English.

IF ... THEN

The original BASIC provided for what has been called a conditional branch. The format starts with the keyword IF, followed by a formula that can be tested internally for a true or false condition, and then the keyword THEN. THEN is followed by a statement to be executed if the statement is true. Two examples:

```
        IF A=B THEN GOTO 10
        IF A+B=C OR IF D+E=F THEN PRINT "END OF PROGRAM"
```

You can use any of the following for a true/false test:

Symbol	Example	Meaning
=	A=B	A is equal to B
<	A<B	A is less than B
<=	A<=B	A is less than or equal to B
>	A>B	A is greater than B
>=	A>=B	A is greater than or equal to B
<>	A<>B	A is not equal to B

If the statement is false, the program continues on to the next line. Internally the test results in a 0 if false and a 1 if true. A formula can be used such as A - 5. If the result is 0 the statement is false and the THEN part is skipped. Any other value is taken as true.

ADDING MACHINE

Let's use an IF ... THEN statement in our adding machine program. In our adding machine program, when the screen gets full, the program stops. We have a keyword SCROLL that will move the entire display up one line allowing us to continue without stopping. We don't want this to happen until we get to the last line on the screen. Here is what we can do. Set up a counter and use an IF statement. When we get to the last line of the screen, we tell the computer to start scrolling.

For the adding machine program the counter is setup on line 100. There are 22 lines on the screen, but because of the headings we choose to start scrolling on the 17th entry.

Program 4-1. Adding Machine

```
 10 REM "ADDER"
 20 REM (C) R. ORRFELT
 30 REM SEPT 1982
110 PRINT "ENTER STARTING BALANCE"
120 INPUT BALANCE
130 PRINT BALANCE
140 PRINT "ENTER AMOUNTS ONE AT A TIME"
150 PRINT
160 PRINT "AMOUNT","BALANCE"
170 PRINT
180 INPUT AMOUNT
190 LET BALANCE=BALANCE+AMOUNT
200 PRINT AMOUNT,BALANCE
210 LET COUNT=COUNT+1
220 IF COUNT >= 17 THEN SCROLL
230 GOTO 180
```

So now we have our simple program with some prompts, heading and a scrolling feature. The program should still be quite easy to read.

FOR ... NEXT

The original BASIC provided for what is called a loop. This lets you repeat a section of the program a fixed number of times. The form is:

```
100 FOR N=2 TO 10 STEP 2
110 PRINT N
120 NEXT N
```

FOR, TO, STEP, and NEXT are all keywords. The above program PRINTs the numbers from 2 to 10 by 2′s. Often the STEP will be a one, in which case the keyword STEP and its size are omitted. Loops can be used within loops. These are called nested loops because they cannot cross:

```
        Allowed             Not allowed

        FOR X=1 TO 8        FOR X=1 TO 8
        FOR Y=2 TO 5        FOR Y=2 TO 5
        NEXT Y              NEXT X
        NEXT X              NEXT Y
```

CLOCK

An excellent example of nested loops is a clock program. Here the loop for seconds is nested inside the loop for minutes and these two loops are nested inside the loop for hours. Such a program is shown in Program 4-1. Let´s digress from BASIC keywords and examine a complete clock program in detail. Program 4-2, the clock program is shown starting on the next page.

The nested loops are as follows:

```
        400 FOR H=B TO 11
        500 FOR M=C TO 59
        600 FOR S=1 TO 60
        730 NEXT S
        750 NEXT M
        780 NEXT H
```

First we want to draw the clock face. Line 200 puts the letter "O" in the center of the face just for looks. Lines 210-230 put the numbers 1-12 in a circle. There are 2*PI radians in a circle. For 12 segments this is 2*PI/12 or PI/6 in this program. This compacts the program.

Next we want to set the starting time. Input B is the hour and input C is the minutes. The trick here is to ENTER the hour and type in the next minute coming up on your watch but don´t press ENTER. Then about 2 seconds before the minute changes press ENTER and the clock program will start off at the right time.

Program 4-2. CLOCK

```
 10 REM "CLOCK"
 20 REM (C) R. ORRFELT
 30 REM AUGUST 1982
100 LET O=0
110 LET P=0
120 LET Q=0
130 LET U=0
140 LET V=0
150 LET X=0
160 LET Y=0
200 PRINT AT 10,15;"O"
210 FOR N=1 TO 12
220 PRINT AT 10-10*COS (N*PI/6),15+10*SIN (N*PI/6);N
230 NEXT N
300 PRINT AT 21,0;"SET HR:"
310 INPUT B
320 PRINT AT 21,0;"SET MIN:"
330 INPUT C
340 PRINT AT 21,0;"        "   [eight spaces]
400 FOR H=B TO 11
410 LET A=H*PI/6
420 UNPLOT P,Q
430 LET P=31+10*SIN A
440 LET Q=22+10*COS A
450 PLOT P,Q
500 FOR M=C TO 59
510 LET A=M*PI/30
520 UNPLOT U,V
530 LET U=31+15*SIN A
540 LET V=22+15*COS A
550 PLOT U,V
600 FOR S=1 TO 60
610 LET A=S*PI/30
620 UNPLOT X,Y
630 LET X=31+18*SIN A
640 LET Y=31+18*COS A
650 PLOT X,Y
660 LET Z=1
```

Program 4-2. CLOCK (cont.)

```
700 FOR I=0 TO 2
705 REM
706 REM
707 REM
710 NEXT I
730 NEXT S
750 NEXT M
760 LET C=0
780 NEXT H
790 LET B=0
800 GOTO 400
```

The position of the hands are P and Q for the hour hand, U and V for the minute hand, and X and Y for the second hand. Before plotting the positon of the hand we erase the old hand by using UNPLOT. Lines 100-160 initialize these numeric variables for the first pass. We calculate the position of the hand and then PLOT it.

Lines 700-707 are the timing adjustment so that the seconds loop takes exactly one second. This is found by trial and error and probably need changing for different machines.

When the minutes loop (59) is completed, line 760 resets the minutes to zero and then advances the hour. A similar thing happens at 12 o'clock. The hour position is set to zero and the clock cycle starts all over again.

Numeric variables (P, Q, U, etc.) must be defined in a LET or INPUT statement before it can be used elsewhere. If you examine lines 420-450 you will notice that we UNPLOT (or erase) the old clock hand positions before we PLOT its new position. On the first pass of the program, these variables would be undefined except for lines 110-160.

This program originally was too large for 2K memory. To compact the program the letter O was set to zero and used whenever a zero was needed.

PRINT

We have looked at inputs, lots of things to put in between, and now let's look at outputs. The keyword is PRINT and can have the following forms:

A number	[PRINT 5]
A numeric variable	[PRINT X]
A string	[PRINT "ENTER A NUMBER"]
A string variable	[PRINT B$]

Following the keyword PRINT we can if we choose control the location where the printing starts. This location is determined by any of the following:

,	[PRINT X, next print at columns 0 or 16]
;	[PRINT X; next print at next position]
AT	[PRINT AT 5,5;X prints X at line 5, column 5]
TAB	[PRINT TAB 8;X prints at column 8]
SCROLL	[SCROLL moves display up and next print at 0,21]

Some comments are in order. You can use PRINT,X to advance the print position before X is printed. Or you can use PRINT,,X to advance the position twice. The semi-colon is required after the column and row for AT and after the column for TAB to hold the print position. You can use a numeric variable or a formula to indicate the column or row. You can also use a series of locations and items after the keyword PRINT.

The PRINT next print position is stored at memory addresses 16441 and 16442. If you are clever, you can use PEEK and POKE to find or alter where you want to print. For an example of this see Program 8-3 (Chapter VIII).

If you loop on TAB, the amount is added on each loop. The maximum total is 255 (one byte) so you may have to set up a counter and readjust the print position for some programs.

DIGITAL CLOCK

After all that, we need to look at a program example. For this we will use Program 4-3, a digital clock that produces large numerals five lines high. Of interest here are the PRINT statements starting on line 1000 that produce the numerals.

For a digital clock we need six digits. The first digit is a special case being either a 1 or nothing. For the other five digits we need five nested loops. Originally:

```
        500 FOR H=1 TO 12
        600 FOR T=0 TO 5
        630 FOR M=0 TO 9
        700 FOR D=0 TO 5
        730 FOR S=0 TO 9
        900 NEXT S
        910 NEXT D
        920 NEXT M
        940 NEXT T
        960 NEXT H
```

Then some of the constants were changed to variables for setting the starting time or for compacting. The result is shown in Program 4-3. The value assigned to X in each loop is the PRINT position across the screen. The numeral to be PRINTed is the loop count i.e. S is seconds and has values from 0 to 9. The GOSUB formula finds the correct PRINT statement. Before examining the PRINT statements, let's look at the first part of the program.

Lines 100-130 define the four strings needed to form parts of the numerals. [g] indicates a graphics mode space so A$ is three black squares. Five C$'s each printed just below the last one, form a 1. These are used in the PRINT statements.

Lines 200-250 define constants needed to compact the program. Lines 310-360 are used to set the starting time. U is the starting tens minutes digit and B becomes the

Program 4-3. Digital Clock

```
 10 REM "D.C"
 20 REM (C)R.ORRFELT
 30 REM AUG 1982
100 LET A$="ggg"    [g ][g ][g ]
110 LET B$="g g"    [g ][ ][g ]
120 LET C$="  g"    [ ][ ][g ]
130 LET D$="g  "    [g ][ ][ ]
200 LET N=8
210 LET O=9
220 LET P=10
230 LET Q=11
240 LET R=12
250 LET K=1000
310 PRINT " INPUT HR AND MIN"
320 INPUT A
330 INPUT B
340 CLS
350 LET U=INT (B/P)
360 LET B=B-P*U
400 PLOT 18,24
410 PLOT 18,20
420 PLOT 38,24
430 PLOT 38,20
500 FOR H=A TO R
510 LET C=H
520 IF C>O THEN GOSUB 2000
530 LET X=5
540 GOSUB K+C*P
600 FOR T=U TO 5
610 LET X=Q
620 GOSUB K+T*P
630 FOR M=B TO O
640 LET X=15
650 GOSUB K+M*P
700 FOR D=0 TO 5
710 LET X=21
720 GOSUB K+D*P
730 FOR S=0 TO O
740 LET X=25
```

Program 4-3. Digital Clock (cont.)

```
 800 FOR Z=1 TO O
 810 NEXT Z
 900 NEXT S
 910 NEXT D
 920 NEXT M
 930 LET B=0
 940 NEXT T
 950 LET U=0
 960 NEXT H
 970 LET A=1
 980 CLS
 990 GOTO 400
1000 PRINT AT N,X;A$,AT O,X;B$,AT P,X;B$,AT Q,X;B$,AT R,X;A$
1005 RETURN
1010 PRINT AT N,X;C$,AT O,X;C$,AT P,X;C$,AT Q,X;C$,AT R,X;C$
1015 RETURN
1020 PRINT AT N,X;A$,AT O,X;C$,AT P,X;A$,AT Q,X;D$,AT R,X;A$
1025 RETURN
1030 PRINT AT N,X;A$,AT O,X;C$,AT P,X;A$,AT Q,X;C$,AT R,X;A$
1035 RETURN
1040 PRINT AT N,X;B$,AT O,X;B$,AT P,X;A$,AT Q,X;C$,AT R,X;C$
1045 RETURN
1050 PRINT AT N,X;A$,AT O,X;D$,AT P,X;A$,AT Q,X;C$,AT R,X;A$
1055 RETURN
1060 PRINT AT N,X;A$,AT O,X;D$,AT P,X;A$,AT Q,X;B$,AT R,X;A$
1065 RETURN
1070 PRINT AT N,X;A$,AT O,X;C$,AT P,X;C$,AT Q,X;C$,AT R,X;C$
1075 RETURN
1080 PRINT AT N,X;A$,AT O,X;B$,AT P,X;A$,AT Q,X;B$,AT R,X;A$
1085 RETURN
1090 PRINT AT N,X;A$,AT O,X;B$,AT P,X;A$,AT Q,X;C$,AT R,X;A$
1095 RETURN
2000 LET C=C-P
2010 LET X=1
2020 GOTO 1010
```

units minutes digit. Lines 400-430 insert two colons at the proper position.

Lines 500-990 are the timing loops for the clock. The hours two digits are unique. Hours from 1 to 9 have a blank for the first digit and only a 1 is needed for the hours of 10 through 12. Line 520 tests for the 1. If C is greater than 9 (P) the program GOSUBs to line 2000 and inserts the 1. At 12:59:59 + one second line 970 resets the hour to 1 and clears the screen. The clock then starts off at 1:00:00. Lines 800-810 adjust the units seconds loop to one second.

Lines 1000-1095 are the PRINT statements to form the digits. There are five ATs on each PRINT line. N, O, P, Q, and R are the screen line locations. X is the column position. A$, B$, C$, and D$ are the black squares used to form the digits.

CHAPTER V

FUN AND GAMES

This chapter is about an arcade game, animation, dice, and a card game. Games and animation are often very machine dependent. The games and animation in this chapter are written for machines with small memories (2K or 3K). If you have more than 3K RAM memory, the display is memory mapped, that is all lines are padded out with spaces to 32 characters. Clearing the screen or rewriting part of the screen is much slower, giving the effect of a shade being drawn from the top down. If you have more than 3K RAM, POKE 16389,76. This limits your active memory to 3K. These programs will then RUN as intended.

The graphic characters and inverse characters (white on black) can be PRINTed just like other characters. To develop a new figure you may wish to use grid paper, or you may wish to "doodle" on the screen using print statements. A good imagination is helpful. For example, to draw an auto, try this:

```
 10 PRINT "  /ggL  "   [ ][ ][/][gs7][gs7][gsL][ ][ ]
 20 PRINT "gggggggg"   [gsH][g ][g ][g ][g ][g ][g ][gsH]
 30 PRINT " O    O "   [ ][O][ ][ ][ ][ ][O][ ]
 g=graphics mode, s=shift,[ ]=1 character
```

PLOT and UNPLOT can also be used as in the Clock program. These two functions are most useful when using functions like SIN and COS to calculate screen positions. All PLOT and UNPLOT graphics can be drawn using the PRINT statement. The choice depends on which is easier to use.

The PAUSE statement causes a disturbance in the display. For most purposes a FOR NEXT loop gives better animation.

```
        10 FOR N=1 TO 37
        20 NEXT N
```

gives about a one second delay. You can use 18 loops for a half second or 370 loops for ten second delay as the delay is proportional.

BEM

The first game is of the arcade type. Here a bug-eyed-monster jumps up at one of ten random positions across the screen. If you press the correct number key you add to your score. You get 20 chances. This game is based on the Sinclair "Sniper" but has been greatly modified.

Refering to Program 5-1, line 90 RAND sets the random number generator to a random starting point. Next the screen is cleared in case the game is played more than once. Two variables are set to zero. S is for your score and C is to count the tries.

This game's key idea is to match the display position to a number keys. Line 130 P=1+INT(RND*10) gives P a random value of from 1 to 10. The function RND generates a number from 0.00000000 to .99999999. Multiplying RND by 10 and taking the INTeger (whole number) is a value of 0 to 9. Adding 1 matches the top row of keys on your computer except we will have a 10 where the 0 is. This is solved on line 140 by multiplying by SGN(10-P). This function has a value of 1 if 10-P is greater than 1, 0 if zero, (and -1 if less than 0). We have now defined K (the key) to correspond to the screen position.

We are going to draw a figure three characters wide for the game, so we multiply P*3 for values from 3 to 30. The last position would try to PRINT at 30, 31, and 32. The maximum position is 31 so we subtract 2 on line 150 to center the display. We have now defined ten positions p and corresponding keys K.

Next we'll need two monster figures, before and after. Let's use an inverse " for the head and eyes and a monster T-shirt (inverse M). Fill in the bodies and PRINT AT the random P position on lines 160 to 190.

To make it more interesting for high scorers, the timing loop on line 200 starts with the score S. This means the higher the score the shorter time the player has to press a key before the monster disappears. Also when a key is pressed, the program jumps out of the timing loop. (If

Program 5-1. "BUG EYED MONSTER"

```
 10 REM "BEM"
 20 REM (C) R. ORRFELT
 30 REM JULY 1982
 90 RAND
100 CLS
110 LET S=0
120 LET C=0
130 LET P=1+INT (RND*10)
140 LET K=INT P*SGN (10-P)
150 LET P=(P*3)-2
160 PRINT AT 4,P;" g "   [ ][g"][ ]
170 PRINT AT 5,P;"ggg"   [gsE][gM][gsR]
180 PRINT AT 6,P;" g "   [ ][g ][ ]
190 PRINT AT 7,P;"ggg"   [gs3][g ][gs4]
200 FOR N=S TO 40
210 IF INKEY$<>""THEN GOTO 400
220 NEXT N
400 IF INKEY$=STR$ K THEN GOTO 500
410 GOTO 800
500 PRINT AT 4,P;"ggg"   [gsY][g"][gsT]
510 PRINT AT 5,P;" g "   [ ][gM][ ]
520 PRINT AT 6,P;" g "   [ ][g ][ ]
530 PRINT AT 7,P;"g g"   [gsT][ ][gsy]
700 LET S=S+1
710 PRINT AT 15,0;"SCORE ";S
720 FOR N=1 TO 30
730 NEXT N
800 LET C=C+1
810 CLS
820 IF C=20 THEN GOTO 900
830 FOR N=1 TO INT(RND*30)+10
840 NEXT N
850 GOTO 130
900 PRINT AT 15,0;"SCORE ";S;
910 PRINT " OUT OF 20"
920 PRINT
930 PRINT "PLAY AGAIN?"
940 INPUT A$
950 IF A$="Y" THEN GOTO 100
```

you are refering to books on other BASICs this may not be allowed but it works OK for Sinclair BASIC. The question is - has anything been left on the machine stack or the GOSUB stack? - this case no).

If the correct key has been pressed, the spread-out monster is displayed or else the count is increased without an increase in the score. A random length pause is inserted before the next monster is displayed. At the end of the game the score is given and the player may choose to play again.

PEEKIN' DUCK

The next item on the agenda is to develop animation. The trick here is to include a space on the side away from the motion to erase the old figure. Let's shoot an arrow across the screen. Try this:

```
10 FOR N=0 TO 28
20 PRINT AT 10,N;"-->";
30 NEXT N
RUN
```

As the arrow advances, the last "-" is left and fills the line. Now add a space at the beginning of the string (" -->"). The arrow now appears to fly across the screen.

Let's do our duck. In fact let's do four ducks; facing right, out and in the water and facing left, in and out of the water. Program 5-2 is the result. The four ducks are constructed by lines 200 through 540. The two ducks facing right begin with at least one space and the two facing left end with at least one space. The six spaces on line 310 erase the top line of the standing duck.

Back to the beginning. After printing the title, lines 100 through 150 draw some land and water. The duck then does its thing. Line 600 lets the duck take a brief nap. We now need something to wake "PD". Lines 700 and 710 makes a "thing" pop up out of the water. (A fish or frog or may-be a monster). After a slight delay the thing pops back under the water by lines 740 and 750. The program repeats as "PD" goes to see what happened.

Program 5-2. "PEEKIN DUCK"

```
 10 REM "PD"
 20 REM (C) R. ORRFELT
 30 REM JULY 1982
 40 PRINT AT 18,2;"PEEKIN DUCK"
 50 PRINT AT 19,2;"BY BOB ORRFELT"
100 FOR N=0 TO 15
110 PRINT AT 15,N;"g"        [gs7]
120 NEXT N
130 FOR N=16 TO 31
140 PRINT AT 15,N;"g"        [gsS]
150 NEXT N
200 FOR N=0 TO 12
210 PRINT AT 12,N;" ,   gg"  [ ][,][ ][ ][ ][g0][gs6]
220 PRINT AT 13,N;"  gggg"   [ ][ ][gsS][g ][g ][gs1]
230 PRINT AT 14,N;"    g"    [ ][ ][ ][ ][gsW]
240 NEXT N
300 FOR N=13 TO 25
310 PRINT AT 12,N;"      "   [ ][ ][ ][ ][ ][ ]
320 PRINT AT 13,N;" ,   gg"  [ ][,][ ][ ][ ][gs0][gs6]
330 PRINT AT 14,N;"  gggg"   [ ][ ][gsS][g ][g ][gs1]
340 NEXT N
400 FOR N=25 TO 14 STEP -1
410 PRINT AT 13,N;"gg   , "  [gs6][g0][ ][ ][ ][,][ ]
420 PRINT AT 14,N;" gggg "   [ ][gs2][g ][g ][gs1]
430 NEXT N
500 FOR N=13 TO 0 STEP -1
510 PRINT AT 12,N;"gg   , "  [gs6][g0][ ][ ][ ][,][ ]
520 PRINT AT 13,N;" gggg  "  [ ][gs2][g ][g ][gsS][ ][ ]
530 PRINT AT 14,N;"  g    "  [ ][ ][gsQ][ ][ ][ ][ ]
540 NEXT N
600 PRINT AT 12,1;"g"        [g-]
610 FOR N=1 TO 20
620 NEXT N
700 PRINT AT 13,29;"?"
710 PRINT AT 14,29;"g"       [g"]
720 FOR N=1 TO 20
730 NEXT N
740 PRINT AT 13,29;" "
750 PRINT AT 14,29;" "
760 GOTO 200
g=graphics mode, s=shifted, [ ]=1 character
```

CRAPS

In this dice game you play against the house (computer). You start with $1000. The program just about fills the 2K memory so there is very little error checking. If you bet all or more than you have, and lose, the game is over. If you wish to practice side bets, you are on your own and must keep your own side bet score.

After you make your bet, the house rolls the dice. The third sides that come up are the dice at rest. If they show a 7 or 11 (natural) you lose. If they show a 2, 3, or 12 (craps) you win. If the result is a 4, 5, 6, 8, 9, or 10 that is the point that the house must make. You win if a 7 comes up before the point is made.

The odds for this game are slightly in favor of the player (252 to 245, or about 1.4%). Casinos change the odds by calling a 2 or a 12 a push or stand-off (neither a win nor loss). This can be done by making the following changes to the program.

```
        330 IF P=3 OR P=12 THEN GOTO 700
        335 IF P=2 THEN GOTO 720
```

This changes the odds to 976 to 949 or about 1.4% in favor of the house. If you wish to automate this program with a fixed bet, make the following change.

```
        230 LET B=100
```

Looking at Program 5-3, line 90 sets the random number seed. Lines 100-140 set the graphics for the dice which are printed by GOSUBs to lines 1010 through 1063. S is your starting amount and B is your bet. Each GOSUB 800 rolls the dice. Lines 310-340 check for a natural or craps, and if not sets the point. If a point is set, the rolls continue until the point is made or a 7 comes up. Line 350 causes the repeat. The use of three rolls is just for graphic effect.

On a loss the score is adjusted, a pause for viewing, clear screen, a check to see if there is money for another bet,

Program 5-3. "CRAPS"

```
 10 REM "CRAPS"
 20 REM (C) R. ORRFELT
 30 REM JULY 1982
 90 RAND
100 LET A$="ggg"     [g0][g ][g ]
110 LET B$="ggg"     [g ][g0][g ]
120 LET C$="ggg"     [g ][g ][g0]
130 LET D$="ggg"     [g0][g ][g0]
140 LET E$="ggg"     [g ][g ][g ]
150 CLS
160 LET S=1000
200 PRINT "YOU HAVE $";S
210 PRINT "BET ";
220 INPUT B
230 PRINT B
300 GOSUB 800
301 GOSUB 800
302 GOSUB 800
310 LET P=D+E
320 IF P=7 OR P=11 THEN GOTO 600
330 IF P=2 OR P=3 OR P=12 THEN GOTO 700
340 PRINT AT 15,0;"MY POINT IS ";P
350 GOSUB 800
351 GOSUB 800
352 GOSUB 800
360 PRINT "ROLLED ";(D+E); " "
400 IF P=D+E THEN GOTO 600
410 IF (D+E)=7 THEN GOTO 700
500 GOTO 350
600 PRINT "  YOU LOSE"
610 LET S=S-B
620 GOSUB 2000
630 CLS
640 IF S<=0 THEN GOTO 900
650 GOTO 200
700 PRINT "  YOU WIN"
710 LET S=S+B
720 GOSUB 2000
730 CLS
740 GOTO 200
```

Program 5-3. "CRAPS" (cont.)

```
800 LET D=1+INT (RND*6)
810 LET E=1+INT (RND*6)
820 LET X=10
830 GOSUB (1000+10*D)
840 LET X=15
850 GOSUB (1000+10*E)
860 RETURN
900 PRINT AT 10,2; "YOU ARE ";
910 PRINT "BROKE"
920 PRINT "PLAY AGAIN?"
930 INPUT G$
940 IF G$="Y" THEN GOTO 150
950 STOP
1010 PRINT AT 8,X;B$
1011 PRINT AT 9,X;B$
1012 PRINT AT 10,X;E$
1013 RETURN
1020 PRINT AT 8,X;A$
1021 PRINT AT 9,X;E$
1022 PRINT AT 10,X;C$
1023 RETURN
1030 PRINT AT 8,X;A$
1031 PRINT AT 9,X;B$
1032 PRINT AT 10,X;C$
1033 RETURN
1040 PRINT AT 8,X;D$
1041 PRINT AT 9,X;E$
1042 PRINT AT 10,X;D$
1043 RETURN
1050 PRINT AT 8,X;D$
1051 PRINT AT 9,X;B$
1052 PRINT AT 10,X;D$
1053 RETURN
1060 PRINT AT 8,X;D$
1061 PRINT AT 9,X;D$
1062 PRINT AT 10,X;D$
1063 RETURN
2000 FOR N=1 TO 60
2010 NEXT N
2020 RETURN
```

and then back to ask for the next bet.

On a win the score is adjusted, a pause, clear screen, and then back to ask for the next bet.

Line 800 is the roll of the dice. D and E are the two die. X is the start of the screen location for the graphics. The GOSUBs calculate the line number of the die display.

When the game ends you are asked if you wish to play again. If you answer by entering "Y" the game starts over. If not, the end.

BLACKJACK

Blackjack is one of the more popular card games. The first task in writing card game programs is to define a deck and to shuffle the deck. Arrays (DIM statements) take up a lot of memory and run slowly. Instead, Program 5-5 uses string slicing to select a card from the deck. A first thought might be to use the first 52 characters of the character set for the string. However, character 11 is the quote and this cannot be used in a string. The solution is Program 5-4 for forming a deck of 52 cards and can be used for other games as well as blakjack.

Program 5-4. "DECK"

```
  10 REM "BJ"
 100 LET A$="1234567890123456789
01234567890123456789012345678901
2"
 110 LET A=16528
 120 FOR N=129 TO 180
 130 POKE A,N
 140 LET A=A+1
 150 NEXT N
```

RUN the above program and then delete lines 110 through 150. You now have the deck needed for the blackjack program.

The shuffle is line 1000 to 1090 of Program 5-5. First a loop of from 1 to 52 is set up. Then a random number from 1 to 52 is generated (X). Then each card in sequence is temporarily stored in B$. A card is next selected at random (A$(X)) and moved to the original location replacing A$(N). This way every card gets moved at least once and is replaced by a random card.

The rules of this blackjack game are as follows. You are playing against the dealer. The game starts with a shuffle of the deck. This takes about 4 seconds. You start with $1000 and must place a bet before the deal. You get two cards face up and the dealer gets one down and one up.

The value of the cards are: Ace, 1 or 11 whichever is best; king, queen, jack, ten, 10 each; any other card, its number. The object is to hold two or more cards that total 21 or as nearly 21 as possible without going over 21. If the player gets a 21 in the first two cards, he wins regardless of the dealer's hand. If the player does not have 21 but the dealer does, the dealer wins.

If neither has 21 then you are asked if you wish a "HIT?". If you enter "Y" you are given another card. If you go over 21 you lose. If not you are asked again if you wish a "HIT?". If you ENTER anything except the "Y" it means you stand. When you stand it is the dealer's turn. The dealer must take cards until he has 17 or more. At this point, if the dealer has more than 21 you win. Otherwise the one with the highest number of points wins. Equal points "push" and neither wins.

The usual player's strategy should be to always stand on 17 or higher. Stand on any number from 13 through 16 if the dealer's card is 7 or higher including an ace. Hit 12 or under. Also if you have 17 or less and one card is an ace, hit.

Because of limited memory, this program does not include splitting doubles or other variations of the game.

Program 5-5 uses lots of code compacting which makes some of the statements obscure. The first four variables starting on line 100 are A$ for the deck, R$ for the rank, S$ for the suit, and V$ for the value. The cards ten and above are given a value of A which is one above 9 and an ace is given the value of B. Line 4060 set V to a value of from 2 to 11.

Lines 140 to 260 initialize the variables except line 180 shuffles the deck. Variables are:

O= zero	E= dealer's PRINT position
I= one	P= player's points
M= dealer's GOSUB line	D= dealer's points
W= the starting $ am't	F= player's aces
Q= player's PRINT position	G= dealer's aces

Lines 300-330 PRINT the players POKE and get the bet.

Lines 400-550 get the player's hand and lines 600-700 get the dealer's hand.

Lines 1000-1070 are the shuffle which has been described.

Lines 2000-2040 get the player's card and count the aces, G. The card is then displayed and Q is set to the next PRINT position.

Lines 3000-3060 get the dealer's card and count the aces, F. Except for the hole card, H$, the card is then displayed and E is set to the next PRINT position.

Lines 4000-4090 get a card from the deck and evaluate it. If the last card has been used, a new deck is sorted. The cards in play will be duplicated in the new deck. C is a counter to develop the suit. X is the position of the card in the deck. M$ is the card formatted for printing.

Lines 5000-5060 adjust the win or loss and after a pause goes back to line 210 for the next hand.

Program 5-5. "BLACKJACK"

```
 10 REM "BJ"
 20 REM (C) R. ORRFELT
 30 REM JULY 1982
 90 RAND
100 LET A$="    deck     "
110 LET R$="A23456789TJQK"
120 LET S$="SDHC"
130 LET V$="B23456789AAAA"
140 LET O=NOT PI
150 LET I=SGN PI
160 LET M=3000
170 LET W=1000
180 GOSUB W
210 LET Q=2
220 LET E=12
230 LET P=0
240 LET D=0
250 LET F=0
260 LET G=0
300 CLS
310 PRINT " POKE $";W,"BET $";
320 INPUT B
330 PRINT B
400 GOSUB 2000
410 GOSUB M
420 GOSUB 2000
430 GOSUB M
440 IF P=21 OR IF D=21 THEN GOTO 5010
460 PRINT " HIT?"
470 INPUT Y$
480 IF Y$<>"Y" THEN GOTO 600
490 GOSUB 2000
500 IF P<22 THEN GOTO 460
510 IF G=0 THEN GOTO 5000
520 LET P=P-10
530 PRINT " ";P
540 LET G=G-I
550 GOTO 460
```

Program 5-5. "BLACKJACK" (cont.)

```
600 PRINT AT E,I;H$;D
605 LET E=E+I
610 IF D<17 THEN GOSUB M
620 IF D<17 THEN GOTO 610
630 IF D<22 THEN GOTO 680
640 IF F=0 THEN GOTO 5030
650 LET D=D-10
655 PRINT " ";D
660 LET F=F-I
670 GOTO 610
680 IF D<>P THEN GOTO 5020
700 GOTO 5040
1000 FAST
1010 FOR N=I TO 52
1020 LET X=I+INT (RND*52)
1030 LET B$=A$(N)
1040 LET A$(N)=A$(X)
1050 LET A$(X)=B$
1060 NEXT N
1070 LET X=I
1080 SLOW
1090 RETURN
2000 GOSUB 4000
2010 LET P=P+V
2015 IF R$(C)="A" THEN LET G=G+I
2020 PRINT AT Q,I;M$;P;
2030 LET Q=Q+I
2040 RETURN
3000 GOSUB 4000
3010 LET D=D+V
3015 IF R$(C)="A" THEN LET F=F+I
3020 LET H$=M$
3030 IF E=13 THEN RETURN
3040 PRINT AT E,I;M$;D;
3050 LET E=E+I
3060 RETURN
```

Program 5-5. "BLACKJACK" (cont.)

```
4000 IF X=53 THEN GOSUB 1000
4005 LET S=I
4010 LET C=CODE A$(X)-128
4020 IF C<=13 THEN GOTO 4060
4030 LET C=C-13
4040 LET S=S+I
4050 GOTO 4020
4060 LET V=CODE V$(C)-28
4070 LET X=X+I
4080 LET M$=R$(C)+" OF "+S$(S)+" "
4090 RETURN
5000 LET B=-B
5010 PRINT AT E,I;H$;D
5020 IF D>P THEN LET B=-B
5030 LET W=W+B
5040 FOR N=I TO 100
5050 NEXT N
5060 GOTO 210
```

CHAPTER VI

INTERNAL AFFAIRS

To write good programs, it is necessary to understand some of the internal workings of the Timex 1000 computer. We can use the computer itself to learn how things work. Two problems all programmers must contend with are speed and limited RAM storage space. Functions do not all take the same amount of time to run. Programmers have many choices of how to write programs. This chapter goes into great detail to help you make these choices.

SPEED

Before getting into the details of what is going on inside the computer, let's look at how fast some of the functions RUN. If we type in the following program, we find it takes about 10 seconds to RUN.

```
10 FOR N=1 TO 360
30 NEXT N
40 PRINT "DONE"
```

This means 36 loops is about 1 second or each loop is a little less than .03 seconds. (Use your Timex 1000 to do the calculations.) Now let's ENTER the functions one at a time and RUN and time them. We will use line 20 to get inside the loop.

```
 20 LET X=2        [.01 seconds]
 20 LET X=2+2      [.016 seconds]  same for +,INT,IF THEN
 20 LET X=SIN .5   [.25 seconds]   same for *,-,/,SIN,RND
 20 LET X=2**2     [.7 seconds]    same for **,SQR
```

So where speed is important, try to avoid using powers (**) and square roots.

The computer is very busy at all times. When it appears to be slow it is because of the many things it needs to do. Without an overall view of what is going on it is easy to get lost in the details of each operation.

Where you do need to make a long computation and need to speed up the program, use the FAST mode. To the previous program you can add:

```
    5 FAST
   50 SLOW
```

HARDWARE

A block diagram of the Timex 1000 is shown in Figure 6-1. The four major integrated circuits that make up most of the hardware are the CPU, ROM, RAM, and the I/O Logic Array. In general their functions are:

CPU (Central Processing Unit) gets an instruction (called an Op Code) from ROM or RAM. An ALU (Arithmetic Logic Unit may perform addition, a compare, etc. in the process.

ROM (Read Only Memory) contains the operating system programs and the BASIC language programs. This is a series of Op codes, ROM and RAM addresses, and data such as the keyboard character codes used for TV display.

RAM (Random Access Memory) stores system variables as needed by the ROM programs, your BASIC program, the TV display character file, and the CPU return stack (SP). The CPU stack is a list of addresses which permit the CPU to get back to the beginning after jumping around in memory.

The I/O Logic Array does the timing needed to generate the TV raster, helps generate the code when a key is pressed, and provides the circuits for interfacing to the tape cassette. These are called I/O ports for the CPU. One other function is performed. Address line A14 and other control signals from the CPU are decoded to select ROM or RAM.

COMPUTER CYCLES

Computers are dynamic devices, i.e. they are always cycling through a logic sequence. While waiting for an external event such as a key being pressed, it is in what is called a loop. Until some condition is true, the sequence jumps

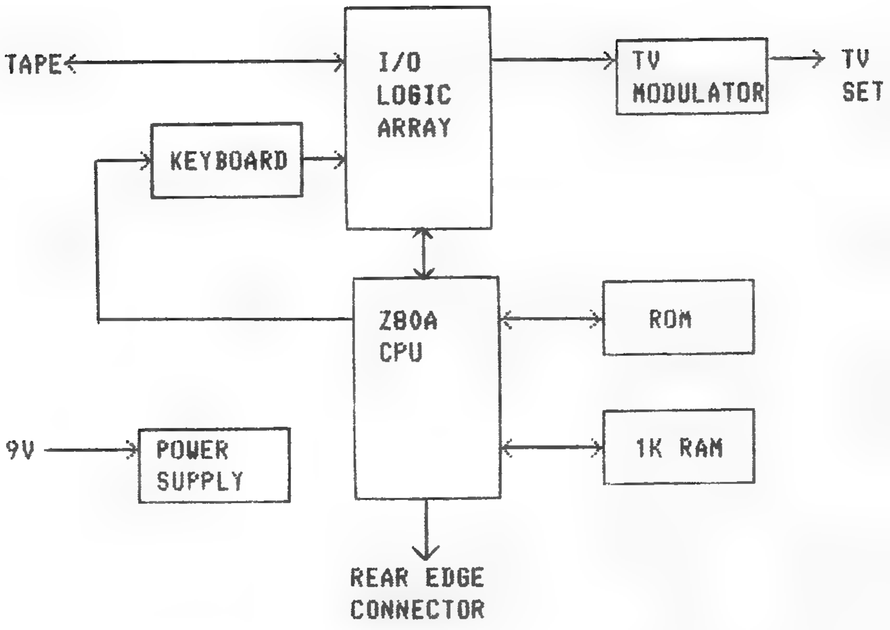


Figure 6-1. Timex 1000 Block Diagram

back to the beginning. These cycles (starting with the fastest) are identified as:

The clock cycle
The machine cycle
The instruction cycle
The system program cycle
A program RUN cycle

THE CPU CLOCK

The CPU clock cycle is always running at 3,250,000 steps per second. The Z80A CPU can operate up to 4 MHz but the exact frequency was chosen for compatibility with the TV display requirements. The CPU clock is used to step the internal CPU logic. An internal logic sequence is called a machine cycle.

THE MACHINE CYCLE

There are three Z80A CPU machine cycles identified as fetch (get the Op code), read, and write. The total number of clock cycles for the three machine cycles makes one instruction cycle.

THE INSTRUCTION CYCLE

An instruction cycle may take from 6 to 33 clock cycles. Each Op code has been given a name (mnemonic) and is the basis for assemby language programming. Chapter VIII has more on Op codes. The point here is that over 100,000 instructions can be handled per second.

THE SYSTEM PROGRAM CYCLES

The system programs refer to those programs that reside in thousands. The beginning (and end) of the system cycle is shown by a message or the keyword prompt in the lower left corner of the screen. In either case the system is scanning the keyboard waiting for a key to be pressed. In the SLOW mode, most of the time is used to generate the TV display leaving only about 12,000 instructions per second for other things. The I/O Logic Array uses an interrupt to ensure the TV timing. While rewriting the display, the computer is now slowed to the point where we begin to see the time it takes for program cycles.

THE PROGRAM RUN CYCLE

When we RUN a BASIC program it may take from a few seconds to many minutes. The time it takes may be programmed deliberately by the use of PAUSE statements or may be the result of scanning the variables file. This brings us to the organization of RAM.

RAM FILES

First, a discussion of how the memory is used and used up and then second, back to how the use of memory affects the time to RUN a program. The 1000 uses a system called dynamic memory allocation. Some of the lower addresses are reserved for use by the ROM programs and contain the addresses of the other RAM files. RAM files include:

RAM System File
Program File
Display File
Program Variables File
Stacks

RAM SYSTEM FILE

Numbers and addresses that change from time to time as needed by the programs in ROM were allocated by the people who wrote the system programs. The Z80A CPU provides some temporary storage but most is in RAM. The first 125 bytes are reserved for this purpose and are called System Variables. You can PEEK at them (and POKE some of then if you are careful) for use in BASIC programs.

PROGRAM FILE

The program file always starts at 16509. The top of the file is just below the start of the display file. Each line contains two bytes for the line number, two bytes for line length, and an ENTER character at the end of the line. The first item must be a keyword followed by the codes for the characters or tokens. After each constant a six byte binary form of the constant is inserted. See Program 6-2.

DISPLAY FILE

The Display File is the memory image of the screen display. This file may take from 25 to 793 bytes of RAM. Whenever the screen is blank, the file consists of just 25 ENTER characters (118). The first ENTER character is the start of the file (see Program 6-2). The following 24 ENTER characters represent the end of each line. If the screen is full, including the bottom two lines normally used for input, the display file will include 768 plus the 25 ENTER characters. Program 6-3 is an expansion of Program 6-1 to show the way the Display File expands as the program is RUN. The top of the display file grows from address 16787 to address 17060 as the program RUNs. By the time the dis-

Program 6-1. System File

PROGRAM

```
 10 REM "SYSTEM FILE"
100 PRINT "SYSTEM";TAB 9;"CONTENT"
110 PRINT "ADDRESS"
120 PRINT
200 FOR N=0 TO 26 STEP 2
210 LET A=16386+N
220 LET B=A+1
230 PRINT A;":    ";
240 PRINT PEEK A+256*PEEK B
250 NEXT N
RUN
```

DISPLAYED

```
SYSTEM    CONTENT
ADDRESS

16386:    18428   [Top of machine stack]
16388:    18432   [Top of RAM + 1]
16390:    61440   [*]
16392:    0       [*]
16394:    110     [CURRENT line number]
16396:    16695   [Start of Display File]
16398:    16815   [Print position]
16400     16844   [Program Variables File]
16402:    16882   [Address of variables]
16404:    16903   [ENTER file location]
16406:    16674   [PEEK or POKE item]
16408:    49176   [*]
16410:    16945   [Calculator stack bottom]
16412:    16959   [End of RAM in use]

[*] content not properly decoded
```

Program 6-2. Program File

PROGRAM

```
 10 REM 23
 20 REM "23"
 30 LET B=1
 40 LET B=B+23
100 FOR N=0 TO 150
110 LET W=PEEK (16509+N)
120 PRINT W;" ";
130 IF W=118 THEN PRINT
257 NEXT N
RUN
```

DISPLAYED

```
        This is line 10
          The line has 4 characters
          REM  2  3 ENTER
0 10 4 0 234 30 31 118

          REM  "  2  3  " ENTER
0 20 6 0 234 11 30 31 11 118


           LET  B  =  1 [   binary 1  ] ENTER
0 30 11 0 241 39 20 29 126 129 0 0 0 0 118


           LET  B  =  B  +  2  3 [   binary 23  ] ENTER
0 40 14 0 241 39 20 39 21 30 31 126 133 56 0 0 0 118


            FOR  N  =  0 [  binary 0 ]  TO  1  5  0
0 100 21 0 235 51 20 28 126 0 0 0 0 0 223 29 33 28
         [   binary 150 ] ENTER
         126 136 22 0 0 0 118


        1 1 3 0 243 51 118
        118
```

play is completed, 8 more bytes will be added for the last address plus 5 more for the message on line 23. The end of RAM in use will then be address 17104. Unused RAM is about 200 bytes for this program.

Let's trace a keyboard entry after NEW. The display file will consist of 24 ENTER characters, an inverse K, and one ENTER. The screen will be blank except for the inverse K in the lower left corner of your screen.

When you press the 1 key, everything is shifted up in memory and a 1 is placed just before the inverse K. The 1 is also placed in the ENTER file. On the next screen scan, the number appears on your TV set.

Now press the P key. Because you are in the keyword mode the word PRINT (with spaces) is placed in the display file and the token for PRINT is placed in the ENTER file. The ENTER file is needed for editing to distinguish between a token and the spelled-out word. The prompt character is changed to an inverse L.

Now press the ENTER key. At the end of the next scan the ENTER file content is formatted and placed in the Program File. The line is then reformatted from the Program File to the first line of the Display File. The line 23 is replaced with just the inverse K character and the ENTER file is erased.

When memory gets full, the display file is the last to go. The display will be chopped-off at the bottom by the dynamic allocation system to prevent loss of other files. When you see this happen, it is a warning that memory is short. During a program listing everything is still OK even if only a few lines are showing. The minimum display is 25 ENTER characters and enough room to display commands and/or messages.

Program 6-3. Display File

PROGRAM

```
 10 REM "DISPLAY FILE"
100 PRINT "SYSTEM";TAB 8;"CONTENT","TOP OF"
110 PRINT "ADDRESS","DISPLAY"
120 PRINT
130 LET C=16400
200 FOR N=0 TO 26 STEP 2
210 LET A=16386+N
220 LET B=A+1
230 PRINT A;":  ";
240 PRINT PEEK A+256*PEEK B,
250 PRINT PEEK C+256*PEEK(C+1)
260 NEXT N
RUN
```

DISPLAYED

```
SYSTEM   CONTENT  TOP OF
ADDRESS           DISPLAY

16383:   18428    16851    [display file top after 18428
16388:   18432    16872     is PRINTed]
16390:   61440    16893
16392:   0        16914
16394:   110      16935
16396:   16765    16956
16398:   16954    16977
16400:   16990    16998
16402:   17041    17019
16404:   17069    17040
16406:   16712    17061
16408:   49214    17082
16410:   17132    17103
16412:   17153    17124    [display file after 17153]
```

PROGRAM VARIABLES FILE

Each time an INPUT, LET, DIM, or FOR/NEXT statement is encountered in your program, as the program is RUN, an entry is made in the variables file. If the variable name already exists it is updated. If it is not found the new variable is created. They remain there until erased by a CLEAR, RUN, or NEW command. There are six formats for storing program variables as follows:

1. Strings (i.e. LET A$="1")
   Takes three bytes plus the number of characters in the string. RUN′s fast. (Four bytes in example).

2. One letter numeric (i.e. LET A=1)
   Always 6 bytes. RUN′s a little slower than a string as it needs conversion to binary.

3. More than one letter numeric (i.e. LET ACE=2)
   This one takes five bytes plus the number of characters in the name. Eight are used in the example. These variables RUN slow as the name is scanned for length when looking for a match.

4. String Arrays (i.e. DIM B$(2,3))
   Takes four bytes plus two times the number of dimensions plus the number of elements. Only 14 bytes are used for the example. RUN′s fairly fast and takes little memory.

5. Numeric Arrays (i.e. DIM B (2,3))
   Takes four bytes plus two times the number of dimensions plus five times the number of elements. In the example 38 bytes are used.

6. FOR/NEXT loop (i.e. FOR N=0 TO 200 STEP 2)
   Always takes 18 bytes. Space for STEP is part of the format.

Program 6-4 displays the variables file showing each type of entry. The first byte of each entry contains the code which is the entry type plus the code of the first char-

Program 6-4. Variables File

PROGRAM

```
 10 REM "VARS FILE"
110 LET A$="1"
120 LET A=1
130 LET ACE=1
140 DIM B$(2,3)
150 DIM B(2,3)
160 FOR N=0 TO 94
170 LET W=PEEK 16400+256*PEEK 16401
180 LET W=W+N
190 PRINT PEEK W;" ";
200 NEXT N
RUN
```

DISPLAYED

```
Var.  1st Byte

A$        70  1 0 29                          [String Variable]
A        102  129 0 0 0 0                     [Numeric Array]
ACE      166  40 170 129 0 0 0 0              [Numeric Array]
B$(2,3)  199  11 0 2 2 0 3 0 0 0 0 0 0        [String Array]
B(2,3)   135  35 0 2 2 0 3 0 0 0 0 0 0
         0 0 0 0 0 0 0 0 0 0 0 0 0 0
         0 0 0 0 0 0 0 0 0 0                  [Numeric Array]
N        243  135 16 0 0 0 135 60 0 0 0 129
         0 0 0 0 161 0                        [FOR/NEXT Loop]
```

Note: 128 (80H) is the end of vars. file character.

acter of the name. For example, A$ is a string and the first byte is 32 + 38 which equals 70. When searching for an entry, only the first byte need be checked for type and name. An exception is numeric variable names with more than one character in the name. In this case additional characters have to be tested for match.

STACKS

There are two stacks at the top of RAM, the GOSUB and the machine stack. The GOSUB stack is the return line number for your BASIC subroutines. The machine stack is a return stack for the Z80 CPU. The GOSUB stack starts with 4 bytes for two line numbers but may vary in length. 70 to 100 or even more may be needed for the stacks.

CHAPTER VII

SLICK TRICKS

This chapter consists of unrelated tips and tricks. The clever programmer can make a computer easy to use, dramatic, and even useful. There are ways to jam more program into 2K memory. There are ways to speed up a program. Who knows what obscure logic lurks in the mind of a computer. Only the programmer.

INKEY$

This function is a real sleeper. During a program run, it tests the keyboard for a single string character. It does not wait for ENTER. First let's look at a pause control (not PAUSE). This subroutine is used in the next chapter in Program 8-3.

```
500
 .
 .
600 IF INKEY$="M" THEN GOTO 700
610 RETURN
700 IF INKEY$="N" THEN RETURN
710 GOTO 700
```

Looking at the keyboard we notice the M key has the word PAUSE just above it. Above the N key is the word NEXT. each line in the main body of the program we use a GOSUB 500. If the M key is not held down, line 610 is a RETURN and the next line is PRINTed. If the M key is held down, we go to line 700 which waits for the N key to be pressed. Used with SCROLL this gives the user control over the display. To the user the keys are PAUSE and NEXT but to the programmer they are M and N.

NEW

This next trick uses both INKEY$ and NEW. If you wish to send a secret message or have a program that only you can

LOAD, just use NEW to blot out the program.

```
 10 REM "NEW"
 20 PRINT "MEET ME AT THE ROUND HOUSE."
 30 PRINT
 40 PRINT "YOU CAN'T CORNER ME THERE."
400 SAVE "NEW"
410 IF INKEY$<>"L" THEN NEW
420 GOTO 20
```

Use GOTO 400 to SAVE the program on tape. When it is LOADed the program will be erased by line 410 if the L key is not held down.

One more word about INKEY$. The characters 5, 6, 7, and 8 have four arrows on them which suggests a display control for games and charts. Characters Q and W are suited for PLOT and UNPLOT control.

24 LINE SCREEN

You can POKE 16418 to use all 24 lines on your TV screen. This byte controls the lines reserved for the ENTER file display. If You POKE in a 0 in a program you can use all 24 lines. Don't forget to POKE in a 2 before the program ends. Try this:

```
10 POKE 16418,0
20 FOR N=1 TO 24
30 PRINT "THIS IS THE LINE NUMBERED ";N
40 NEXT N
50 POKE 16418,2
60 PAUSE 1000
```

If you wish to protect the lower line of the screen from scrolling, you can POKE in a 3. A poke in the right place does wonders.

TV SCREEN TEST

You can use your computer to test focus, linearity, and convergence. The following program uses the POKE 16418,0 to display a full screen of dots.

```
        10 POKE 16418,0
        20 FOR N=0 TO 383
        30 PRINT "..";
        40 NEXT N
        50 IF INKEY$="" THEN GOTO 50
        60 CLS
        70 POKE 16418,2
```

There is INKEY$ again on line 50. This freezes the display until a key (any key) is pressed.

COMPACTING A PROGRAM

Consider the statement: LET X=0. In the program file, the 0 takes one byte followed by its binary form which takes six more bytes. An equivalent but obscure form would be: LET X=NOT PI where NOT and PI are functions. NOT is a logic function which says that PI is not a zero and therefore the expression is false. A 0 is assigned to X to show the statement is false. Anyway, the second form takes two bytes of program storage compared to seven for the first form, a saving of five bytes. However, the display file is five bytes longer while the program is being displayed. Usually this is not a problem as the program variables can be cleared while listing the program.

Here are three short forms of constants:

```
        NOT PI = 0
        SGN PI = 1
        INT PI = 3
```

Remember the X is added to the program variables file and that takes six more bytes. If a variable is used as a constant in only one or two places, it is shorter to use the constant in each statement. If a variable can be reused in different parts of a program, six bytes are saved each time. Also a FOR/NEXT loop creates an eighteen byte variable. If it can be reused eighteen bytes are saved.
If at the end of a program you wish to start over and don't need to save the variables, use RUN instead of GOTO 10. This saves eight bytes.

Tokens can be used inside quotes to save space. For keywords type the token for THEN, then the keyword, shift over the keyword with the left arrow, and then delete the word THEN. Tokens for functions can be typed directly.

String variables can be used to compact print statements. Here is a program called "A$;C$".

Program 7-1. A$,C$.

```
 10 REM "A$;C$"
 20 LET A$=" WOOD"
 30 LET B$=" WOULD "
 40 LET C$="CHUCK "
 50 PRINT "HOW MUCH";A$;B$;"A";A$;C$" ";C$
 60 PRINT
 70 PRINT "IF A";A$;C$;B$;C$;A$;"?"
 80 PRINT AT 20,0;
 90 PRINT "PRESS ""CONT"" FOR ANSWER"
100 STOP
110 CLS
120 PRINT "A";A$;C$;B$;C$;"AS MUCH";A$;"AS A";A$;C$;B$
130 PRINT "IF A";A$;C$;B$;C$;A$;"."
```

CLUB DEMO

Program 7-2 is a moving display for putting in a store window to get attention of passersby. You can change the messages to suit your own purposes. The subroutine on lines 1010 to 1030 cause the entire screen to scroll up and off the screen. If you GOSUB 1000 you get a few second delay before scrolling. The main part of the program has four parts:

Part 1. Lines 100-230 draw a box and put the main theme inside.

Part 2. Lines 240-490 first put up a sign and then has a puppy go to it for a look. Puppy wags tail.

Program 7-2. Club Demo.

```
 10 REM "DEMO"
 20 REM (C) R. ORRFELT
 30 REM OCT. 1982
100 FOR N=9 TO 53
110 PLOT N,36
120 PLOT N,6
130 NEXT N
140 FOR N=36 TO 6 STEP -1
150 PLOT 9,N
160 PLOT 53,N
170 NEXT N
200 PRINT AT 8,11;"JOIN THE"
210 PRINT AT 10,11;"TS  1000"
220 PRINT AT 12,11;"CLUB NOW"
230 GOSUB 1000
240 FOR N=1 TO 30
250 PRINT AT 13,N;"g"    [gsS]
260 NEXT N
300 PRINT AT 10,22;"MAKE NEW"
310 PRINT AT 11,22;"FRIENDS"
400 FOR N=0 TO 14
410 PRINT AT 10,N;" g  gg"  [ ][gs4][ ][ ][gs)][gs4]
420 PRINT AT 11,N;" gggg"   [ ][gS8][g ][g ][gs5]
430 PRINT AT 12,N;"  g g"   [ ][ ][gsW][ ][gsW]
440 NEXT N
450 FOR N=0 TO 100
460 PRINT AT 10,16;" "
470 PRINT AT 10,16;"g"    [gs4]
480 NEXT N
490 SCROLL
```

(continued)

Part 3. Lines 500-590 add messages to the screen.

Part 4. Lines 600-620 list the end of the program. Don't list more than the last 21 lines or you will get an error message which will stop the display. Note the message starting at 900.

Line 630 then causes the entire process to repeat.

BINARY DUMP

Program 7-3 is a binary dump. If you are a student of machine code, you can enter any address and see the binary codes displayed with the address shown every eight bytes. Three of the more interesting addresses are given. The dot matrix pattern for the character set that is used by the TV display begins at the address 7680. If you wish to skip to the zero or the letter A, use the other addresses shown. Stop the display by using the BREAK key.

Line 280 PRINTs either a 0 or an inverse 1 for emphasis. If you wish just 0's and 1's change the line to:

```
        280 PRINT D
```

Binary representation is formed by dividing a byte C by descending binary numbers 128, 64, 32, 16, 8, 4, 2, and 1 and taking the integer which will be a 0 or 1.

Program 7-2. Club Demo (cont.)

```
 500 PRINT "EVERY MONDAY NIGHT"
 510 SCROLL
 520 SCROLL
 530 PRINT "SHARE YOUR PROGRAMS"
 540 SCROLL
 550 SCROLL
 560 PRINT "SIGN UP HERE"
 570 SCROLL
 580 SCROLL
 590 GOSUB 1000
 600 CLS
 610 LIST 900
 620 GOSUB 1000
 630 GOTO 100
 900 REM ** YOU CAN LEARN BASIC
 910 REM     PROGRAMMING **
 920 REM
 930 REM
1000 GOSUB 2000
1010 FOR N=1 TO 22
1020 SCROLL
1030 NEXT N
1040 RETURN
2000 FOR N=1 TO 100
2010 NEXT N
2020 RETURN
```

Program 7-3. Binary Dump

```
 10 REM "BDUMP"
 20 REM (C) R. ORRFELT
 30 REM OCT 1982
100 PRINT "BINARY DUMP"
110 PRINT
120 PRINT "ADDRESS","FOR"
130 PRINT
140 PRINT "7680","""" """"
150 PRINT "7904","""0"""
160 PRINT "7984","""A"""
170 PRINT
180 PRINT "ENTER FIRST ADDRESS:"
190 INPUT A
200 SCROLL
210 PRINT A
220 SCROLL
230 FOR J=1 TO 8
240 LET C=PEEK A
250 LET B=128
260 FOR N=1 TO 8
270 LET D=INT (C/B)
280 PRINT CHR$ (28+129*D);
290 LET C=C-B*D
300 LET B=B/2
310 NEXT N
320 SCROLL
330 LET A=A+1
340 NEXT J
350 GOTO 210
```

CHAPTER VIII

WHAT THE HEX?

Who needs hexadecimal numbers? First, there are those who are interested in the ROM program. Disassemblers print the programs in hex. For this you will need two more books; one with the disassembled ROM and one on the Z80 CPU instruction set. Second, there are those who wish to do programming in machine code. This is usually used to control special I/O devides so you'll need to buy or to build hardware. A Z80 CPU book on the instruction set will also be needed. This chapter just offers some tools needed to get started.

The conversion from hex to binary is given here as a reference.

HEX		BINARY		DECIMAL	HEX		BINARY		DECIMAL
0h	=	0000	=	0	8h	=	1000	=	8
1h	=	0001	=	1	9h	=	1001	=	9
2h	=	0010	=	2	Ah	=	1010	=	10
3h	=	0011	=	3	Bh	=	1011	=	11
4h	=	0100	=	4	Ch	=	1100	=	12
5h	=	0101	=	5	Dh	=	1101	=	13
6h	=	0110	=	6	Eh	=	1110	=	14
7h	=	0111	=	7	Fh	=	1111	=	15

Hexadecimal numbers are the bridge between binary, machine code, and hardware. What you need to develop is double think. When you see address 4000h you should think binary 0100 0000 0000 0000. Address lines are numbered from A0 to A15. 4000h tells you A14 is high (about four volts) and all other address lines are low (about 0.5 volts). Line A14 is decoded to turn the ROM off and to turn RAM on. So 4000h is the first address in RAM. Maybe this should be called double double think.

Each memory address is accessed by eight data lines. This stores eight bits or one byte. Bytes are machine Op codes or other data as required. A byte is represented by two hex numbers.

DECIMAL TO HEX

Program 8-1 converts decimal numbers from 0 to 65535 to hex format. Decimal 65535 is FFFFh. Decimal 0 is 0000h. The program works by counting the number of 4096's, 256's, 16's and the remainder. Counting is done by lines 310 to 420 in the program. X is the most significant figure, then Y, then Z, and C is the remainder. The characters 0 through F are formed by adding 28 to the value for printing. Another way of doing the same thing is shown in Program 8-3, lines 520 through 570. The counting method is used here as it is easier to understand.

Program 8-1. Decimal to Hex Conversion

```
 10 REM "DTOH"
 20 REM (C) R. ORRFELT
 30 REM SEPT 1982
100 SCROLL
110 PRINT "ENTER DECIMAL NUMBER"
120 SCROLL
200 INPUT B
210 LET X=0
220 LET Y=0
230 LET Z=0
300 LET C=B
310 LET C=C-4096
320 IF C>=0 THEN LET X=X+1
330 IF C>=0 THEN GOTO 310
340 LET C=C+4096
350 LET C=C-256
360 IF C>=0 THEN LET Y=Y+1
370 IF C>=0 THEN GOTO 350
380 LET C=C+256
390 LET C=C-16
400 IF C>=0 THEN LET Z=Z+1
410 IF C>=0 THEN GOTO 390
420 LET C=C+16
430 SCROLL
440 PRINT B,CHR$ (X+28);CHR$ (Y+28);
450 PRINT CHR$ (Z+28);CHR$ (C+28);"H"
500 GOTO 200
```

HEX TO DECIMAL

Program 8-2 converts hex numbers from 0 to FFFFh to decimal numbers. The key idea in this program is to evaluate each character in the hex number by position. String slicing is used in lines 296 to 299 to select one character. The number of characters, L, is found on line 210. A GOTO line is calculated by 300-L to go to the line having the correct value. For five digits you could add a line:

```
        295 LET N=65536*(CODE A$(L-4)-28)
```

and so on. Both this and the previous program illustrate the value of each position in a hex number.

Program 8-2. Hex to Decimal Conversion

```
 10 REM "HTOD"
 20 REM (C) R. ORRFELT
 30 REM SEPT 1982
100 SCROLL
110 PRINT "ENTER HEX NUMBER"
120 SCROLL
200 INPUT A$
210 LET L=LEN A$
220 LET N=0
230 GOTO 300-L
296 LET N=4096*(CODE A$(L-3)-28)
297 LET N=N+256*(CODE A$(L-2)-28)
298 LET N=N+16*(CODE A$(L-1)-28)
299 LET N=N+CODE A$(L)-28
300 SCROLL
310 PRINT A$;"H",N;"D"
500 GOTO 200
```

READING HEX NUMBERS

The conversions of hex numbers from 0 to Fh was given on page 8-1. For larger numbers think as follows. 1 less than 10h is Fh. 1 less than 100h is FFh. 1 less than 1000h is FFFh. 1 less than 10000h is FFFFh. The width of your TV display is 20h characters (2*16). Two lines is 40h, three 60h and so on. Five lines hold A0h characters (remember Ah is 10 decimal). Seven lines is D0h and eight is 100h which is 256 characters. For sixteen lines you will have 200h characters or 512 decimal. 22 lines is found by adding six lines to 16 lines or 2C0h. It is well to remember 0,2,4,6,8,A,C, and E are even numbers in hex. 256 bytes is sometimes called a block. There are four blocks in a 1K memory which is 400 hex. Of course your 2K memory is 800h. Getting to the fourth digit which has has a decimal value of 4096, the 8K ROM holds 2000h bytes. Addresses range from 0000h to 1FFFh. RAM starts at 16K which is 4000h. RAM addresses therefore are from 4000h to to 47FFh. If you add a 16K RAM to your computer, the RAM addresses will be 4000h to 7FFFh (one less than 8000h).

If you look at the character set you will notice a pattern between normal and graphic mode characters. A space is 0 and a graphic space is 80h. A zero is 1Ch and a graphic zero is 9Ch. In each case only the most significant bit is changed which is adding 80h.

HEX DUMP

Enter the starting hex address and this program dumps ROM or RAM bytes in hex. The address of the first byte and then eight bytes temporarily stop the dump, press the M key dump, press the M key (PAUSE); to restart, press the N key (NEXT).

The key idea in this program is in lines 210 and 220. The byte is defined as C and will always be between 00h and FFh. The first character is formed by dividing C by 16 and discarding the remainder. 28 is the CODE for zero and is added to the offset. The second character is formed from the remainder (C-16*INT (C/16)) plus the offset. The next address is A=A+1 and the cycle is repeated.

Starting from the top, the input is stored in A$. Lines 140 and 150 pad A$ out to four characters with leading zeros. The decimal address is formed on line 160. The GOSUB 500 converts the decimal address back to hex and

Program 8-3. Hex Dump

```
 10 REM "DUMP"
 20 REM (C) R. ORRFELT
 30 REM SEPT 1982
100 SCROLL
110 PRINT "ENTER STARTING HEX ADDRESS"
120 SCROLL
130 INPUT A$
140 IF LEN A$<4 THEN LET A$="0"+A$
150 IF LEN A$<4 THEN GOTO 140
160 LET A=4096*(CODE A$(1)-28)+256*(CODE A$(2)-28)
    +16*(CODE A$(3)-28)+(CODE A$(4)-28)
170 GOSUB 500
200 IF PEEK 16441<5 THEN GOSUB 500
210 LET C=PEEK A
220 PRINT CHR$ (28+INT(C/16));CHR$ (28+C-16*INT(C/16));" ";
230 LET A=A+1
240 GOTO 200
500 SCROLL
510 LET B=A
520 FOR N=4 TO 1 STEP -1
530 LET D=INT (B/16)
540 LET A$(N)=CHR$ (B-16*D+28)
550 LET B=D
560 NEXT N
570 PRINT A$
580 SCROLL
600 IF INKEY$="M" THEN GOTO 700
610 RETURN
700 IF INKEY$="N" THEN RETURN
710 GOTO 700
```

Hex Dump Display

```
ENTER STARTING HEX ADDRESS


407D
00 0A 08 00 EA 0B 29 3A 32 35
4087
0B 76 00 14 0F 00 EA 10 28 11
4091
37 1B 00 34 37 37 2B 2A 31 39
409B
```

prints them with scrolling. Line 200 checks the printing position and 5 characters before the end of the line starts a new line. Lines 600-710 are the pause and next routine.

The display shows the beginning of the program itself which always begins at 407Dh. The bytes are now all in hex. The second byte shown is 0Ah which is line 10 of the program. The ENTER character at the end of each line is 76h.

HEX PEEK OR POKE

Program 8-4 lets you enter hex code programs into RAM. When you RUN this program, you are asked for the mode: P for peeking, O for poking, or E for ending the RUN. Next you are asked for the starting address which can be from one to four hex characters. The hex address is A$ and subroutine 1000 forms the decimal address A. If you select P, addresses and hex bytes are displayed until some key is pressed which returns you to the start. If you select O, you are asked for the starting address, and then you can enter hex bytes, one at a time. These bytes must be two characters from 00 to FF. Enter a character larger than FF (say L) and you will be returned to the start.

For peeking, line 310 sets C to the byte at address A. Line 320 prints the hex address A$ and the hex version of C. GOSUB 2000 forms the next address, both the decimal A and the hex A$.

For poking, line 520 gets your input byte. Line 540 displays the hex form and line 550 puts it in memory.

Program 8-4. Hex Peek or Poke

```
  10 REM "PORP"
  20 REM (C) R. ORRFELT
  30 REM SEPT 1982
 100 SCROLL
 110 PRINT "P=PEEK, O=POKE, OR E=END: ";
 120 INPUT P$
 130 PRINT P$
 140 IF P$="E" THEN GOTO 10000
 150 SCROLL
```

Program 8-4. Hex Peek or Poke (cont.)

```
 160 PRINT "ENTER HEX ADDRESS: ";
 170 INPUT A$
 180 PRINT A$
 190 SCROLL
 200 SCROLL
 210 GOSUB 1000
 220 IF P$="O" THEN GOTO 500
 300 REM (HEX PEEK)
 310 LET C=PEEK A
 320 PRINT A$, CHR$ (28+INT (C/16));
     CHR$ (28+C-16*INT (C/16));"H "
 330 GOSUB 2000
 340 IF INKEY$<>"" THEN GOTO 100
 350 GOTO 310
 500 REM (HEX POKE)
 510 DIM B$(2)
 520 INPUT B$
 530 IF B$>"FF" THEN GOTO 100
 540 PRINT A$, B$;"H "
 550 POKE A,(16*(CODE B$(1)-28)+CODE B$(2)-28)
 560 GOSUB 2000
 570 GOTO 520
1000 IF LEN A$<4 THEN LET A$="0"+A$
1010 IF LEN A$<4 THEN GOTO 1000
1020 LET A=4096*(CODE A$(1)-28)+256*(CODE A$(2)-28)+
     16*(CODE A$(3)-28)+(CODE A$(4)-28)
1030 RETURN
2000 SCROLL
2010 LET A=A+1
2020 LET H=A
2030 FOR I=4 TO 1 STEP -1
2040 LET D=INT(H/16)
2050 LET A$(I)=CHR$ (H-16*D+28)
2060 LET H=D
2070 NEXT I
2080 RETURN
```

USR EXAMPLE

As an example of the USR function, we will write a machine code program for printing the RUN number. Because this program does not use a program variable, the RUN number is not erased by RUN and can be saved on tape. We will use the unused byte at 407B (see Table 8-1). Here is the program:

```
3A 7B 40    ld a,407B    [load accumulator with byte at 407B]
3C          inc a        [increment accumulator]
32 7B 40    ld 407B,a    [load 407B byte from accumulator]
4F          ld c,a       [load c register from accumulator]
AF          xor a        [zero accumulator]
47          ld b,a       [load b register with zero]
C9          ret          [return to BASIC program]
```

To proceed, first LOAD program 8-4. Next modify the program by changing line 20 to:

```
        20 REM 12345678901
```

This reserves 11 bytes for our hex program. We will need the RAM address of the first 1 on lilne 20. To get this:

```
        RUN    [run the program]
        P      [for peek]
        407D   [for starting address of program file]
```

Starting at 408E you will see the hex code for 1, 2, 3, 4 (1D 1E etc.). Press any key to stop the listing and get back to the selection mode.

So 408E is the starting address for our machine code. To continue, we enter the hex code as follows:

```
        O      [for poke]
        408E   [starting address in REM statement]
        3A 7B 40 3C 32 7B 40 4F AF 47 C9   [hex code]
```

To end the poke mode, press any key larger than "FF" (say L). The mode selection will be displayed. ENTER E to end the RUN mode. We'll need the decimal version of the hex address so ENTER the following without line numbers:

```
        LET A$="408E"
        GOSUB 1000
        PRINT A
```

The decimal address 16525 will be printed. Now complete the program by entering:

```
        30 SCROLL
        40 PRINT "RUN NO. ";USR 16525
```

Now test the machine code program by RUNning and ending the program a few times verifying the run number is updated each time. SAVE the program on tape, LOAD, and RUN showing the run number is not reset. For other programs, you may just want to run the USR program without needing the number in bc. A popular form is to use: RAND USR (address).

And that is how to use USR!

HEX FIND

Program 8-5 is a utility for finding a sequence of hex bytes of any length. Each time a match is found the address of the first byte is printed. Enter the starting and ending addresses in hex. Next enter the bytes for which you are looking in the following format:

```
        01 3F A4 FF
```

In this program, line 420 tests for a match of a memory byte and D$ byte. D$ is the string of bytes formed on line 320. The GOSUB on line 310 converted your input to one byte per byte. When a match is formed, the FOR/NEXT loop on line 410 is completed and the GOSUB 3000 prints the address.

The GOSUB is used to convert the hex input addresses to binary form. Lines 200-230 juggle A's and B's so that only one subroutine is needed. In the end A is the binary form of A$ and B is the binary form of B$.

Program 8-5. Hex Find

```
 10 REM "FIND"
100 PRINT "ENTER FIRST HEX ADDRESS: ";
110 INPUT A$
120 PRINT A$;"H"
130 PRINT "ENTER LAST HEX ADDRESS: ";
140 INPUT B$
150 PRINT B$;"H"
160 PRINT "ENTER HEX CODE: ";
170 INPUT C$
180 PRINT C$
190 PRINT
200 GOSUB 2000
210 LET A=B
220 LET A$=B$
230 GOSUB 2000
290 LET N=1
300 LET D$=""
310 GOSUB 1000
320 LET D$=D$+CHR$ C
330 IF N<LEN C$ THEN GOTO 310
400 LET A=A-1
410 FOR X=1 TO LEN D$
420 IF PEEK (A+X)<>CODE D$(X) THEN GOTO 450
430 NEXT X
440 GOSUB 3000
450 LET A=A+1
460 IF A=B THEN GOTO 10000
470 GOTO 410
1000 LET C=16*(CODE C$(N)-28)+CODE C$(N+1)-28
1010 LET N=N+3
1020 RETURN
2000 LET B=4096*(CODE A$(1)-28)+256*(CODE A$(2)-28)+
     16*(CODE A$(3)-28)+CODE A$(4)-28
2010 RETURN
3000 LET H=A+1
3010 FOR I=4 TO 1 STEP -1
3020 LET D=INT(H/16)
3030 LET H$(I)=CHR$ (H-16*D+28)
3040 LET H=D
3050 NEXT I
3060 PRINT H$;"H   ";
3070 RETURN
```

CHAPTER VII MENU

Program 8-6 is a tape menu program for this chapter. It uses string slicing to form the name of the program to be loaded. All program names in this chapter are four characters long. A$ is defined as a string of all names run together. When you input a number at line 200, lines 210-220 pick out the corresponding letters from the string.

Program 8-6. Chapter 8 Menu.

```
 10 REM "O"
 20 REM (C) R. ORRFELT
 30 REM SEPT. 1982
100 LET A$="DTOHHTODDUMPPORPFIND"
110 PRINT "SELECT FILE BY NUMBER"
120 PRINT
130 PRINT "1. ";A$(1 TO 4);" DECIMAL TO HEX"
140 PRINT "2. ";A$(5 TO 8);" HEX TO DECIMAL"
150 PRINT "3. ";A$(9 TO 12);" HEX DUMP"
160 PRINT "4. ";A$(13 TO 16);" HEX PEEK OR POKE"
170 PRINT "5. ";A$(17 TO 20);" FIND"
200 INPUT N
210 LET N=N*4-3
220 LET A$=A$(N TO N+3)
230 LOAD A$
240 GOTO 10000
250 SAVE "O"
260 GOTO 100
```

RAM ADDRESSES

Table 8-1 identifies some of the RAM addresses in both hex and decimal. More detail can be found in the 1000 instruction manual supplied with your computer. An example of how to use this information is shown on line 200 of Program 8-3. The column count is checked to prevent overflow of the line. You´ll need to master Chapter VI to make full use of this table.

ROM ADDRESSES

Table 8-2 identifies some of the ROM routines. The DUMP program can be used to display the hex code. The first eight items are "RESTART" addresses which can be called either by a program instruction or by hardware interrupt. The I/O chip puts out an interrupt to restart the TV screen display. For those who wish greater detail, a book on the ZX81 disassembled ROM is recommended.

Table 8-1. RAM Addresses

Hex	Decimal	Comments
4000	16384	Report code byte
4001	16385	System flags
4002	16386	GOSUB return address
4004	16388	System top of RAM address + 1
4006	16390	System mode (K, L, F, or G) byte
4007	16391	Current line number for BASIC (2 bytes)
4009	16393	ROM type byte
400A	16394	Current line number for editor (2 bytes)
400C	16396	Display file address
400E	16398	PRINT position
4010	16400	Program variables file address
4012	16402	Current variable address
4014	16404	ENTER input file address
4016	16406	Next BASIC character address
4018	16408	Previous BASIC character address

(continued next page)

Table 8-1. RAM Addresses (cont.)

Hex	Decimal	Comments
401A	16410	Bottom of stack
401C	16412	End of stack
401E	16414	Calculator's b register
401F	16415	Calculator's scratch pad address
4021	16417	Not used
4022	16418	Number of display lines reserved for input
4023	16419	Top program line for LIST
4025	16421	Last key pressed
4027	16423	Keyboard debounce status
4028	16424	Blank lines between TV frames (31)
4029	16425	Next program line address in RUN
402B	16427	Line number for CONT
402D	16429	System flags
402E	16430	String length (2 bytes)
4030	16432	Next item in syntax table
4032	16434	RND seed. Set by RAND
4034	16436	Frame counter (used by PAUSE)
4036	16438	Last PLOT column
4037	16439	Last PLOT row
4038	16440	Used by LPRINT
4039	16441	Column number for PRINT
403A	16442	Line number for PRINT
403B	16443	Flags
403C	16444	Printer buffer (33 bytes)
405D	16477	BASIC's calculator memory
407B	16507	Not used
407D	16509	Start of Program file
47FF	18431	Top of 2K RAM

Table 8-2. ROM Addresses

Hex	Decimal	Comments
0000	0	Op Code C7, RST 0 Start-up
0008	8	Op Code CF, RST 8 Error reports
0010	16	Op Code D7, RST 10 Print
0018	24	Op Code DF, RST 18 Read BASIC program byte
0020	32	Op Code E7, RST 20 "
0028	40	Op Code EF, RST 28 Floating point (jump 199C)
0030	48	Op Code F7, RST 30 Variables spaces
0038	56	Op Code FF, RST 38 I/O TV raster
0066	102	Non-maskaable I/O (TV) display
007E	126	Keyboard character CODEs
00CC	204	Keyboard function CODEs
0111	273	Keyboard graphics mode CODEs
01FC	508	Called by LOAD and SAVE
0207	519	TV display
02BB	699	Keyboard scan
02F6	758	SAVE command
0340	832	LOAD command
03CB	971	Top of RAM and machine stack
03E5	997	Initialize RAM, cursor, LIST
063E	1598	BASIC interpreter
07B4	1972	Keyboard interpreter from 2BB address
07F1	2033	Used by print RST 10
08F5	2293	Inserts spaces in TV display (2K RAM)
0A2A	2602	CLS command
0ACF	2767	PRINT command
0BAF	2991	PLOT and UNPLOT commands
0C0E	3086	SCROLL command
0C29	3113	BASIC tables
0CBA	3258	BASIC interpreter (see 063E)
0F20	3872	FAST command
0F2B	3880	SLOW command
0F52	3922	Numeric conversion
131D	4893	LET command
1405	5125	DIM command
14CA	5322	Floating point routine
1914	6420	Function table
199C	6550	Floating point
1AA9	6825	Function calculations
1E00	7680	Character dot patterns
1FFF	8191	Top of ROM

APPENDIX A

This appendix consists of a set of tape programs with a description of each program. The following nine programs are included:

These programs are intended as an exercise for learning your computer. Type in the program listings and SAVE them on tape. You will quickly learn the keyboard, editor, SAVE, and LOAD commands. Study the description of each program that is given before each program and look for each key idea.

Type in the MENU first but don't RUN it until after saving all programs. You will also learn the advantages and disadvantages of menus. The tape menu is easy to use but takes considerable time to LOAD a program if it is near the end of a tape with many programs. You may wish to use the cassette microphone to record the name of the program on tape just prior to saving it. You can then use the fast mode and rewind to locate the file.

"MENU" PROGRAM.

Description:

A MENU is a list of programs displayed on the screen for your selection. It is logical to place it at the beginning of the cassette tape. This program:

-Tells you the programs on the tape.
-Allows you to select a program by number.

Procedure:

To load, display, and use a menu perform these steps.

1. Type in LOAD "" but don't press ENTER as yet.
2. Start the cassette in the "play" mode.
3. Press ENTER
4. As words appear on the screen, stop the tape.
5. From the list displayed, select the number of the program you wish.
6. Start the cassette again in the "play" mode.
7. Press ENTER.
8. When the stable display reappears, stop the tape.

Remarks:

The "key" idea is on line 370. When you INPUT a number, it is assigned to A$ and automatically loads the program.

Line	10	the program name for loading is "0"
	100-250	is just a show-off title
	110	a PAUSE 50 is about one second
	120-150	FOR NEXT loop draws top and bottom of box
	130	CHR$ 134 is the + character
	160-190	FOR NEXT loop draws sides of box
	250	clears the screen
	260-350	displays the menu
	360	waits for a number to be entered
	370	is for auto load of selected program
	400	if MENU program is saved using GOTO 400, the program is run automatically when loaded.

Program A-0. "MENU"

```
 10 REM "0"
 20 REM (C) R. ORRFELT
 30 REM JULY 1982
100 PRINT "I AM YOUR "
110 PAUSE 50
120 FOR I=7 TO 23
130 PRINT AT 15,I;CHR$ 134
140 PRINT AT 6,I;CHR$ 134
150 NEXT I
160 FOR I=6 TO 15
170 PRINT AT I,7;CHR$ 134
180 PRINT AT I,23;CHR$ 134
190 NEXT I
200 PAUSE 50
210 PRINT AT 10,13;"ZX81"
220 PAUSE 50
230 PRINT AT 19,23;"HI THERE"
240 PAUSE 50
250 CLS
260 PRINT "SELECT FILE BY NUMBER: "
270 PRINT
280 PRINT "1 CALCULATOR",
290 PRINT "5 LOAN TABLE"
300 PRINT "2 LENGTHS",
310 PRINT "6 BAR CHART"
320 PRINT "3 VOLUMES",
330 PRINT "7 LARGE NUMERALS"
340 PRINT "4 WEIGHTS",
350 PRINT "8 SIGN OFF"
360 INPUT A$
370 LOAD A$
380 STOP
400 SAVE "0"
410 GOTO 20
```

"CALCULATOR" PROGRAM.

Description:

This program simulates the functions of a hand calculator. It operates on the last value (starting with 0) and displays the input and new value. The screen fills and then scrolls as new calculations are displayed at the bottom line. The alphabet is locked-out (except M for memory). Pressing the ENTER key sets the memory to the last value. Pressing the M key retrieves the memory value.

Functions include: +,-,*,**,/,PI,SIN,COS,TAN,ASN,ACS,ATN, LN,EXP,INT, and SQR.

Procedure: Load and RUN this program.

1. ENTER a number without a function for a new series.
2. ENTER a function (+,-,*,**,or /) followed by a value. Use a number or an expression using any function(s).
3. Pressing the ENTER key sets memory to the last total. The M key can now be used to "recall" the memory value.

Remarks:

The "key" idea is line 190 which checks for the function.

Line 100	prints the title
110	spaces one line
120	initializes memory to zero
130	initializes value to zero
140	initializes string to "START"
160	display line
170	input statement
180	checks for "ENTER-only" entry (sets memory)
190	checks for +,-,*,**,or / functions
200	error check for non-valid input
210	gets the first character in string entry and converts function to value (E+E$)
230	gets next input after displaying results
500	sets memory to present value
510	displays memory value
520	gets next input

Program A-1. "CALCULATOR"

```
 10 REM "1"
 20 REM (C) R. ORRFELT
 30 REM JULY 1982
100 PRINT TAB 10;"CALCULATOR"
110 PRINT
120 LET M=0
130 LET V=0
140 LET A$="START"
150 FOR L=2 TO 21
160 PRINT (L-2);" ";A$,V
170 INPUT A$
180 IF A$"" THEN GOTO 500
190 LET C=CODE A$
200 IF C>37 AND C<50 OR C>50 AND C<64 THEN GOTO 170
210 IF C>20 AND C<25 OR C=216 THEN LET A$=STR$ V+A$
220 LET V=VAL A$
230 NEXT L
240 CLS
250 GOTO 150
500 LET M=V
510 PRINT "M= ";M
520 GOTO 230
```

"LENGTHS" PROGRAM.

Description:

This program converts one unit of length to another. The two that follow are similar in structure. You can do other conversion programs using these as a guide. These programs let you select the unit by number.

Procedure:

1. LOAD "2" and RUN the program.
2. ENTER the number for the units you have.
3. ENTER the number for the units you want.
4. ENTER the first value you want to convert.
5. Continue entering values for conversion until the screen is full.
6. ENTER STOP and then RUN for more conversions.

Remarks:

The "key" idea is line 450 where M converts to millimeters and X converts to the units you want.

Lines	100-180	define A$
	200-280	print selection list
	300-330	input "from" units (A)
	340-350	input "to" units (B)
	360-370	input first amount
	400	calculate GOSUB line X
	420	save first conversion factor (M)
	430-440	get next conversion factor (X)
	450	calculate and print conversion
	460	get the next input
	502-512	conversion factors

Program A-2. "LENGTHS"

```
 10 REM "2"
100 DIM A$(8,12)
110 LET A$(1)="INCHES "
120 LET A$(2)="FEET "
```

Program A-2. "LENGTHS" (cont.)

```
130 LET A$(3)="YARDS "
140 LET A$(4)="MILES "
150 LET A$(5)="MILLIMETERS "
160 LET A$(6)="CENTIMETERS "
170 LET A$(7)="METERS "
180 LET A$(8)="KILOMETERS "
200 PRINT "1 ";A$(1),"5 ";A$(5)
210 PRINT "2 ";A$(2),"6 ";A$(6)
220 PRINT "3 ";A$(3),"7 ";A$(7)
230 PRINT "4 ";A$(4),"8 ";A$(8)
300 PRINT "SELECT CONVERSION"
310 PRINT "FROM: ";
320 INPUT A
330 PRINT A$(A);" TO ";
340 INPUT B
350 PRINT A$(B)
360 PRINT "INPUT AMOUNTS"
370 INPUT C
400 LET X=2*A+500
410 GOSUB X
420 LET M=X
430 LET X=2*B+500
440 GOSUB X
450 PRINT C;" ";A$(A);"=";C*M/X;" ";A$(B)
460 GOTO 370
502 LET X=25.4
503 RETURN
504 LET X=304.8
505 RETURN
506 LET X=914.4
507 RETURN
508 LET X=1609344
509 RETURN
510 LET X=1
511 RETURN
512 LET X=10
513 RETURN
514 LET X=1000
515 RETURN
516 LET X=1000000
517 RETURN
```

"VOLUMES" PROGRAM.

Description:

This program converts one unit of volume to another. You can select the units by number.

Procedure:

1. LOAD "3" and RUN the program.
2. ENTER the number for the units you have.
3. ENTER the number for the units you want.
4. ENTER the first value you want to convert.
5. Continue entering values for conversion until the screen is full.
6. ENTER STOP and then RUN for more conversions.

Remarks:

The "key" idea is line 450 where M converts to cubic centimeters and X converts to the units you want.

```
Lines 100-180 define A$
      200-280 print selection list
      300-330 input "from" units (A)
      340-350 input "to" units (B)
      360-370 input first amount
      400     calculate GOSUB line X
      420     save first conversion factor (M)
      430-440 get next conversion factor (X)
      450     calculate and print conversion
      460     get the next input
      502-512 conversion factors
```

Program A-3. "VOLUMES"

```
 10 REM "3"
100 DIM A$(8,12)
110 LET A$(1)="TEASPOONS "
120 LET A$(2)="TABLESPOONS "
```

Program A-3. "VOLUMES" (cont.)

```
130 LET A$(3)="CUPS"
140 LET A$(4)="PINTS "
150 LET A$(5)="QUARTS "
160 LET A$(6)="GALLONS
170 LET A$(7)="CUBIC CMS "
180 LET A$(8)="LITERS "
200 PRINT "1 ";A$(1),"5 ";A$(5)
210 PRINT "2 ";A$(2),"6 ";A$(6)
220 PRINT "3 ";A$(3),"7 ";A$(7)
230 PRINT "4 ";A$(4),"8 ";A$(8)
300 PRINT "SELECT CONVERSION"
310 PRINT "FROM: ";
320 INPUT A
330 PRINT A$(A);" TO ";
340 INPUT B
350 PRINT A$(B)
360 PRINT "INPUT AMOUNTS"
370 INPUT C
400 LET X=2*A+500
410 GOSUB X
420 LET M=X
430 LET X=2*B+500
440 GOSUB X
450 PRINT C;" ";A$(A);"=";C*M/X;" ";A$(B)
460 GOTO 370
502 LET X=4.928
503 RETURN
504 LET X=14.784
505 RETURN
506 LET X=236.5625
507 RETURN
508 LET X=473.125
509 RETURN
510 LET X=946.25
511 RETURN
512 LET X=3785
513 RETURN
514 LET X=1
515 RETURN
516 LET X=1000
517 RETURN
```

"WEIGHTS" PROGRAM.

Description:

This program converts one unit of weight to another. You can select the units by number.

Procedure:

1. LOAD "4" and RUN the program.
2. ENTER the number for the units you have.
3. ENTER the number for the units you want.
4. ENTER the first value you want to convert.
5. Continue entering values for conversion until the screen is full.
6. ENTER STOP and then RUN for more conversions.

Remarks:

The "key" idea is line 450 where M converts to grams and X converts to the units you want.

Lines	100-180	define A$
	200-280	print selection list
	300-330	input "from" units (A)
	340-350	input "to" units (B)
	360-370	input first amount
	400	calculate GOSUB line X
	420	save first conversion factor (M)
	430-440	get next conversion factor (X)
	450	calculate and print conversion
	460	get the next input
	502-512	conversion factors

Program A-4. "WEIGHTS"

```
 10 REM "4"
100 DIM A$(8,12)
110 LET A$(1)="GRAINS "
120 LET A$(2)="OUNCES "
```

Program A-4. "WEIGHTS" (cont.)

```
130 LET A$(3)="POUNDS "
140 LET A$(4)="TONS "
150 LET A$(5)="MILLIGRAMS "
160 LET A$(6)="GRAMS "
170 LET A$(7)="KILOGRAMS "
180 LET A$(8)="METRIC TONS "
200 PRINT "1 ";A$(1),"5 ";A$(5)
210 PRINT "2 ";A$(2),"6 ";A$(6)
220 PRINT "3 ";A$(3),"7 ";A$(7)
230 PRINT "4 ";A$(4),"8 ";A$(8)
300 PRINT "SELECT CONVERSION"
310 PRINT "FROM: ";
320 INPUT A
330 PRINT A$(A);" TO ";
340 INPUT B
350 PRINT A$(B)
360 PRINT "INPUT AMOUNTS"
370 INPUT C
400 LET X=2*A+500
410 GOSUB X
420 LET M=X
430 LET X=2*B+500
440 GOSUB X
450 PRINT C;" ";A$(A);"=";C*M/X;" ";A$(B)
460 GOTO 370
502 LET X=.06481
503 RETURN
504 LET X=28.35
505 RETURN
506 LET X=453.6
507 RETURN
508 LET X=9.072E5
509 RETURN
510 LET X=.001
511 RETURN
512 LET X=1
513 RETURN
514 LET X=1000
515 RETURN
516 LET X=10E6
517 RETURN
```

"LOAN TABLE" PROGRAM.

Description:

This program constructs a mortgage loan amortization table for an existing loan. Required input information is the total amount of the loan, interest rate, the monthly payment amount, and the starting month of the loan. The interest, credit (reduction of principal), and balance owed are printed for each month. Total interest for the year is printed for income tax purpose.

Procedure:

1. Have the required information ready (above) and LOAD and RUN the program.
2. For "INPUT STARTING AMOUNT: $" ENTER the principal of the loan.
3. For "INTEREST RATE: " ENTER the annual interest rate i.e. for 12% enter 12.
4. For "PAYMENTS: $" ENTER the regular monthly payment amount.
5. For "FIRST MONTH: " ENTER the number of the month (1 through 12).
6. The first years table will be displayed. Enter "CONT" for each of the following years.
7. The last payment will be a "baloon" payment.

Remarks:

The "key" calculation is line 250. Interest is rounded to the next highest penny as is the loan industry standard.

Line	100-210	input prompts
	220&340	X is total interest
	300	N is current month
	310	J is current interest
	320	R is credit (amount applied to loan)
	330	B is balance
	500-520	displays the final payment and interest total for the last year.

Program A-5: "LOAN TABLE"

```
 10 REM "5"
 20 REM (C) R. ORRFELT
 30 REM JULY 1982
100 PRINT "INPUT STARTING AMOUNT: $";
110 INPUT B
120 PRINT B
130 PRINT "INTEREST RATE: ";
140 INPUT I
150 PRINT I
160 PRINT "PAYMENTS: $";
170 INPUT P
180 PRINT P
190 PRINT "FIRST MONTH: ";
200 INPUT M
210 PRINT M
220 LET X=0
230 PRINT "MO. INT.";TAB 13;"CREDIT";
    TAB 22;"BALANCE",,,
240 FOR N=M TO 12
250 LET J=INT ((B*I/12)+.5)
260 LET R=P-(J/100)
270 LET B=B-R
280 IF B<R THEN GOTO 500
300 PRINT N;TAB 3;
310 PRINT J/100;TAB 13
320 PRINT R;TAB 22;
330 PRINT B
340 LET X=X+(J/100)
350 NEXT N
400 PRINT
410 PRINT "TOTAL INTEREST: $";X
420 PRINT
430 LET M=1
440 PRINT "ENTER ""CONT"" FOR NEXT YEAR"
450 STOP
460 GOTO 220
500 PRINT N;TAB 3;J/100
510 PRINT "LAST PAYMENT: $";B+R+J/100
520 PRINT "TOTAL INTEREST: $";X+J/100
```

"BAR CHART" PROGRAM.

Description:

Bar charts can be used to show annual trends to a large group. This program allows you to ENTER data one month at a time. Use this to support a lecture.

Procedure:

1. LOAD "6" and RUN this program.
2. The program automatically prints the scales.
3. ENTER an amount from 0 to 20 for each month, one month at a time.

Remarks:

The "key" idea is line 340 that calculates and prints the bar segments.

Line	100-130	prints a vertical scale from 0 to 20
	200-250	using string slicing prints the first letter of the twelve months.
	300	loops for 12 inputs
	310	ENTER amount one month at a time
	320-400	plot the amount
	410	get the next month input

Program A-6. "BAR CHART"

```
 10 REM "6"
 20 REM (C) R. ORRFELT
 30 REM JULY 1982
100 PRINT "20",,,,"18",,,,"16",,,,
110 PRINT "14",,,,"12",,,,"10",,,,
120 PRINT " 8",,,," 6",,,," 4",,,,
130 PRINT " 2",,,," 0"
200 DIM A$(12)
210 LET A$="JFMAMJJASOND"
220 FOR A=1 TO 12
230 PRINT AT 21,(2*A+3);
240 PRINT A$(A)
250 NEXT A
300 FOR A=1 TO 12
310 INPUT B
320 LET C=20-B
330 FOR D=C TO 19
340 PRINT AT (D+1),(2*A+3);
350 IF D=C THEN PRINT "g"   [gs6]
360 IF D=C THEN GOTO 400
370 PRINT "g"               [g ]
400 NEXT D
410 NEXT A
```

"LARGE NUMERALS" PROGRAM.

Description:

The purpose here is to create large numerals for viewing by a large group of people. Characters are five times normal height. Up to eight characters can be displayed, all in-line. Characters are:

0,1,2,3,4,5,6,7,8,9,.,:, and a space.

```
The letter 4 is I=1 B$ = 0 0
                I=2 B$ = 0 0
                I=3 A$ = 000
                I=4 D$ =   0
                I=5 D$ =   0
```

Procedure:

1. LOAD "7" and RUN this program.
2. ENTER up to eight characcter i.e. 12:30:00
3. The characters will be displayed.

Remarks:

The "key" idea here is line 260 which calculates the GOSUB line numbers. For example, if N="0" the value is 0. Times 10 is still zero. The "0" subroutine starts at line 500. F$ is the four characters to be printed, top and bottom is a bar; otherwise print a block, space, block, space.

```
Lines 100-130 define the segments needed
      140-200 E$ is input characters
      210-310 two loops I(1 to 5) = segment,
                        N(1 to 8) = character.
      500-592 0 through 9 segment generators
      600     space character
      700     period
      800     semicolon
```

Program A-7. "LARGE NUMERALS"

```
 10 REM "7"
 20 REM (C) R. ORRFELT
 30 REM JULY 1982
100 LET A$="ggg "   [g ][g ][g ][ ]
110 LET B$="g g "   [g ][ ][g ][ ]
120 LET C$="g   "   [g ][ ][ ][ ]
130 LET D$="  g "   [ ][ ][g ][ ]
140 DIM E$(8)
150 LET E$="        "
160 PRINT AT 10,10;
200 INPUT E$
210 FOR I=1 TO 5
220 FOR N=1 TO 8
230 IF E$(N)=" " THEN GOTO 600
240 IF E$(N)="." THEN GOTO 700
250 IF E$(N)=":" THEN GOTO 800
260 LET G=500+(VAL E$(N)*10)
270 GOSUB G
280 PRINT AT (I+10),(N*4-4);
290 PRINT F$
300 NEXT N
310 NEXT I
320 STOP
500 LET F$=A$
501 IF I=1 THEN RETURN
502 IF I=5 THEN RETURN
503 LET F$=B$
504 RETURN
510 LET F$=D$
511 RETURN
520 LET F$=C$
521 IF I=4 THEN RETURN
530 IF I=1 THEN LET F$=A$
531 IF I=2 THEN LET F$=D$
532 IF I=3 THEN LET F$=A$
533 IF I=4 THEN LET F$=D$
534 IF I=5 THEN LET F$=A$
535 RETURN
```

Program A-7. "LARGE NUMERALS" (cont.)

```
540 LET F$=B$
541 IF I<=2 THEN RETURN
542 LET F$=A$
543 IF I=3 THEN RETURN
544 GOTO 510
550 LET F$=C$
551 IF I=2 THEN RETURN
552 GOTO 530
560 LET F$=C$
561 IF I=2 THEN RETURN
562 GOTO 580
570 LET F$=A$
571 IF I=1 THEN RETURN
572 GOTO 510
580 IF I=1 THEN LET F$=A$
581 IF I=2 THEN LET F$=B$
582 IF I=3 THEN LET F$=A$
583 IF I=4 THEN LET F$=B$
584 IF I=5 THEN LET F$=A$
585 RETURN
590 LET F$=D$
591 IF I=4 THEN RETURN
592 GOTO 580
600 LET F$="    "
610 GOTO 280
700 LET F$=D$
710 IF I=5 THEN GOTO 280
720 GOTO 600
800 LET F$=" g  "  [ ][g ][ ][ ]
810 IF I=2 OR I=5 THEN GOTO 280
820 GOTO 600
900 STOP
```

"SIGN OFF" PROGRAM.

This program demonstrates the use of nested loops (FOR NEXT statements). For the 13 star flag or for the 50 star flag the graphics don't work out too well. However, the 48 star flag divides easily into just a few patterns which can be repeated. The patterns could be divided in different ways. For this program the first six lines of the flag are divided into three groups - a line of stars and dark stripe and then the next line of stars. This is repeated three times. Next four alternating dark and light stripes are drawn. Finally the pole is drawn over the last light stripe leaving the 7 dark and 6 light stripes.

Procedure:

Just LOAD "8" and RUN this program. You don't have to do anything.

Remarks:

The "key" idea here are the loops nested within a larger loop.

Line	10	"8" is for loading from the menu
	100	top of flag pole
	110	C is the outside loop counter
	120	A is the counter for "*"
	150	B is the counter for a stripe
	190-210	is the same as 120-140 (a GOSUB could be used)
	240	C is the counter for stripes at the bottom
	250	A is the counter for the stripe width
	280	sets the pointer to the next line
	290	a white stripe
	310	prints the flag pole over last white stripe
	320	B is the flag pole length
	350	positions sign-off
	360	dramatic pause
	400	freeze display until a key is pressed

Program A-8. "SIGN OFF"

```
 10 REM "8"
 20 REM (C) R. ORRFELT
 30 REM JULY 1982
100 PRINT "g"    [gsH]
110 FOR C=1 TO 3
120 FOR A=1 TO 8
130 PRINT "*";
140 NEXT A
150 FOR B=1 TO 20
160 PRINT "g";    [gsA]
170 NEXT B
180 PRINT
190 FOR A=1 TO 8
200 PRINT "*";
210 NEXT A
220 PRINT
230 NEXT C
240 FOR C=1 TO 4
250 FOR A=1 TO 28
260 PRINT "g";    [gsA]
270 NEXT A
280 PRINT
290 PRINT
300 NEXT C
310 PRINT AT 14,0;
320 FOR B=1 TO 8
330 PRINT "g"     [gsH]
340 NEXT B
350 PRINT AT 20,20;
360 PAUSE 50
370 PRINT "THE END"
400 IF INKEY$="" THEN GOTO 400
```

GLOSSARY

ABS - Absolute value. (Function under the G key) changes a negative number to positive. Positive numbers unchanged.

ADDRESS - A number from 0 to 65535 used by the CPU to select a storage area of RAM or ROM.

BIT - Smallest element of information storage. Represented by a 1 or 0.

BINARY - A numbering system using bits. For example, a binary 1111 1111 1111 1111 is 65535 in decimal.

BYTE - Eight bits. The Z80A CPU moves information one byte at a time. Each memory address stores one byte.

CHARACTER SET - Numbers, letters, and symbols that can be displayed on the TV screen. It is stored in the ROM.

CLEAR - Keyword command erases numeric and string variables and the display file. The program file is not erased.

CLS - Keyword command which erases the display file only.

CODE - The one byte number of the first character in a string. Must be followed by a string (i.e. A$).

CONSTANT - A decimal number when typed in a program. May be plus or minus.

COMMAND - One of the words on the keyboard which control the computer. Except for ENTER they can be used as statements in this computer.

COMPARE - Test for equality. Result is true or false.

CPU - Central Processing Unit. The Z80A CPU microprocessor.

GATE - Hardware logic element.

DIM - Keyword for dimension. Used to reserve space in variables file for numbers or strings.

EXPRESSION - A formula or function which results in a number. A logical expression results in a 1 or 0.

FILE - One of the areas in memory used to store information. Also information stored on tape.

FIRMWARE - Programs which cannot be changed from the keyboard (ROM).

HEX - Short for hexadecimal (16) numbering system.

INT - Integer. Whole number such as needed for counting.

INTERRUPT - An event external to the Z80A causing the program to go to a subroutine (used in machine code).

INVERSE VIDEO - White characters on black background. Used by graphics mode and for prompts.

KEYWORD - One key entry of BASIC word to start a statement or after THEN.

MACHINE CODE - A program written byte for byte as required by the CPU (Z80A).

MODULATOR - Hardware device that converts display to TV channel 2 or 3.

NUMERIC VARIABLE - The name of the place in the RAM variables file where a number is stored.

RAM - Random Access Memory. The 2K memory chip inside the computer or the 16K RAM pack.

REBOOT - Computer term for turning power off and then on to start over.

REGULATOR - A device that maintains a steady 5V for computer circuits from the 9V supply.

ROM - Read Only Memory. The ROM holds the BASIC language and other programs needed to make the computer operate.

SOFTWARE - Programs and data that may be saved on tape or loaded into RAM memory.

STATEMENT - One line in a BASIC program. Starts with a BASIC keyword and followed by required information.

STRING - A sequence of characters representing only themselves.

STRING SLICING - A method of specifying part of a string.

STRING VARIABLE - The name of the place in the variables file where a string is stored.

SUBROUTINE - A short program used by a main program. Used by entering GOSUB and the starting line number.

TOKEN - One byte representation of a word used internally by the computer. Keywords, commands, and function names use tokens.

TRUE - Test resulting in a number other than 0.

Z80A -Microprocessor. Hardware device used by the 1000.

INDEX

ERRATA
HIGHFALUTIN′ COMPUTIN′

Make the following changes as indicated:

On page 4-6 add line 100:
```
    100 LET COUNT=0
```

On page 4-8 change line 640 to read:
```
    640 LET Y=22+18*COS A
```

On page 5-5 add a space at the end of line 420:
```
    420 PRINT AT 14,N;" gggg " [ ][gs2][g ][g ][gs1][ ]
```

On page 5-8 change line 1010 to read:
```
    1010 PRINT AT 8,X;E$
```

On page 5-12 cross out the extra IF on line 440:
```
    440 IF P=21 OR D=21 THEN GOTO 5010
```

On page 8-10 add the following line:
```
    90 DIM H$(4)
```

On page A-5 line 180 should read:
```
    180 IF A$="" THEN GOTO 500
```

On page A-11 correct line 516:
```
    516 LET X=1E6
```

On page A-13 add a ";" at the end of line 310:
```
    310 PRINT J/100;TAB 13;
```

$9.95